VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Navyšování výkonu, migrace a zálohování rozsáhlých informačních systémů

Boost Performance, Migration, and Back-Up of Large-scale Information Systems

Bc. Martin Jagoš

Diplomová práce

Vedoucí práce: Ing. Radoslav Fasuga, Ph.D.

Ostrava, 2021

Abstrakt

Tato diplomová práce se zaměřuje na problematiku zálohování, migrace a navyšování výkonu databázových systémů. Cílem práce je poskytnou čtenáři přehled o možnostech zálohování, navyšování výkonu a vysvětlení procesu migrace a implementovat vlastní řešení. Obsahem první části je analýza technologií virtualizace, kontejnerizace, cloudových služeb a CDN. Další kapitoly se věnují popisu enterprise edicí, postupu zálohování, replikace, migrace databázových systémů a navyšování výkonu. Druhá část práce obsahuje popis použitých technologií a implementace nástrojů.

Klíčová slova

zálohování; navyšování výkonu; migrace

Abstract

This thesis focuses on backup, migration and boost performance of database systems. The goal is to provide the reader with an overview about possibilities for backup, boost performance and migration process and implementation of own solution. The content of first part is analysis of virtualization, containerization, cloud services and CDN. Another chapters are about enterprise editions of databases, backup procedures, replication, migration of database systems and boost performance. Second part contains description of used technology and implementation.

Keywords

backup; performance boost; migration

Poděkování

Rád bych poděkoval Ing. Radoslavu Fasugovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování diplomové práce.

Obsah

Seznam použitých zkratek a symbolů

IS	–	Informační systém
CDN	–	Content Delivery Network
SaaS	–	Software as a Service
PaaS	–	Platform as a Service
IaaS	–	Infrastructure as a Service
AWS	–	Amazon Web Services
TDE	–	Transparent Data Encryption
OLTP	–	Online Transaction Processing
TLS	–	Transport Layer Security
SSL	–	Secure Sockets Layer
SNMP	–	Simple Network Management Protocol
LDAP	–	Lightweight Directory Access Protocol
SIEM	–	Security information and Event Management
LVM	–	Logical Volume Manager
HTTP	–	Hypertext Transfer Protocol
SQL	–	Structured Query Language
CPU	–	Central Processing Unit
SSD	–	Solid-State Drive
HDD	–	Hard Disk Drive
APT	–	Advanced Packaging Tool
JSON	–	JavaScript Object Notation
CSV	–	Comma-separated values

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Databáze jsou nedílnou součástí informačních systémů, protože slouží k ukládání a zpracování dat, která jsou základem pro fungování všech těchto systémů. Běžně mohou s IS pracovat desítky až tisíce uživatelů v libovolných denních i nočních hodinách, a proto musí být data neustále dostupná a zabezpečena proti ztrátě. Z těchto důvodů musí správci databází nasadit správný proces zálohování a replikace pro rychlé řešení náhlého výpadku serveru a zamezení ztráty velkého množství dat. Vysoké nároky jsou kladeny také na výkon informačního systému a správci tak musí provádět navyšování výkonu databáze pomocí různých technik a nástrojů, případně migrovat databázi na vhodnější databázovou technologii nebo výkonnější server.

Cílem této práce je analyzovat a zdokumentovat možnosti zálohování, migrace a navyšování výkonu informačních systémů a následně navrhnout postupy zálohování a migrace a demonstrovat je na praktických příkladech. V oblasti navyšování výkonu je cílem implementovat nástroj, který by sloužil jako pomocník a nabízel by doporučené kroky pro zlepšení výkonu databáze. Posledním úkolem je porovnat implementované nástroje a postupy s existujícími projekty.

První část se bude zabývat teoretickým zpracováním diplomové práce. Nejprve budou popsány technologie virtualizace, kontejnerizace, cloudových služeb a CDN, které lze v rámci řešení použít. Následně se budeme zabývat analýzou a popisem různých způsobů zálohování a replikace vybraných databázových systémů, vysvětlíme proces migrace a popíšeme možnosti navyšování výkonu. Druhá část bude zaměřena na použité technologie a knihovny a dále na popis implementace jednotlivých nástrojů. Na závěr budou zmíněny možnosti rozšíření implementovaných nástrojů a srovnání dosažených výsledků s referenčními projekty.

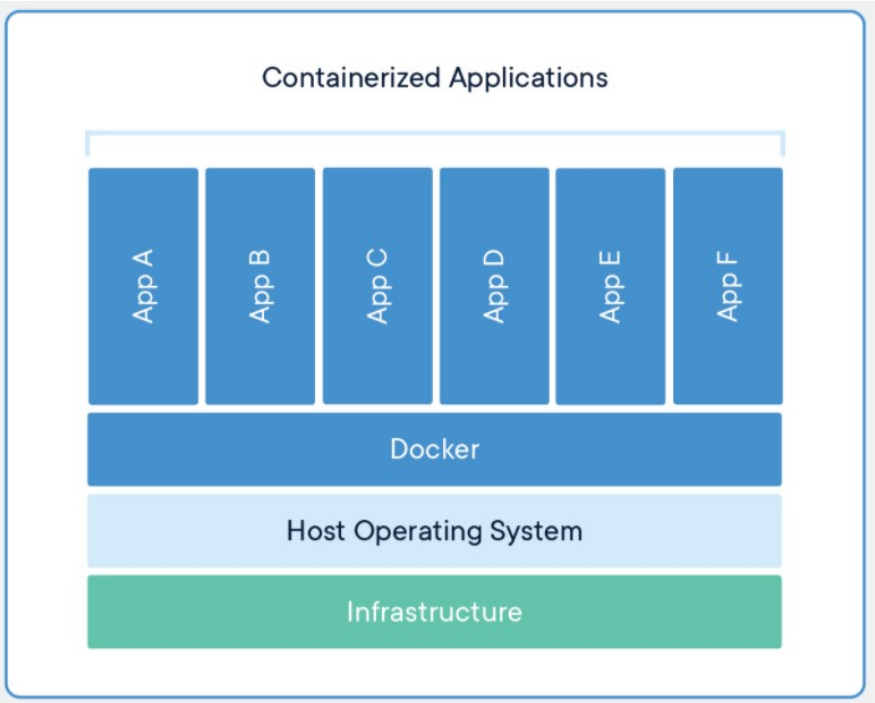
Kapitola 2

Popis aktuálního stavu

V této kapitole je obsaženo shrnutí znalostí z literatury a internetových zdrojů týkající se problému této diplomové práce. Tento problém se týká informačních systémů, zejména jejich migrací, zálohování a navyšování jejich výkonu. Pro řešení tohoto problému se zaměříme hlavně na využití kontejnerových služeb (software Docker), virtualizace, cloudových služeb a CDN (Content delivery network). Po nastudování literatury z internetových zdrojů týkající se výše uvedených přístupů k řešení problému vznikly následující podkapitoly. Ty nás seznámí s jejich možnostmi pro řešení migrace, zálohování, obnovy a navyšování výkonu.

2.1 Kontejnerové služby

Kontejnerové služby jsou prvním přístupem, kterým se budeme zabývat jako jedním z možných přístupů pro řešení diplomové práce. Zřejmě první otázka, která musí být zodpovězena pro pochopení kontejnerů a jejich využití pro řešení našeho problému je, co kontejner v informatice vlastně znamená. Je to standardizovaná jednotka software, která zapouzdřuje kód a všechny jeho závislosti. Díky tomu aplikace běží stejně rychle a spolehlivě na jakémkoliv výpočetním prostředí [1]. Kontejnerizace se také nazývá odlehčenou virtualizací. Přesněji řečeno se jedná o virtualizaci operačního systému. Naše kontejnery tedy mohou běžet v rámci jednoho operačního systému a sdílí jeho jádro, ale běží jako izolované procesy [1]. Nejznámějším software pro tvorbu a správu kontejnerů je nástroj Docker. V roce 2018 podle společnosti Sysdig běželo 83% kontejnerů právě pomocí této aplikace [2]. Docker také nabízí repozitář kontejnerových obrazů, pomocí kterého může komunita dockeru nebo vývojové týmy sdílet vytvořené kontejnery. Za zmínku také stojí nástroj LXC, CoreOS rkt nebo Apache MESOS. Hlavním případem užití kontejnerizace je překonání problému rozdílného vývojového a produkčního prostředí, nemusíme se tedy při vývoji zaobírat rozdílnou platformou nebo hardware [3]. Dalším případem užití jsou například testovací účely, kdy chceme otestovat novou verzi software, ale nechceme provést aktualizaci. V porovnání s virtuálními stroji, kontejnerizované aplikace využívají méně paměti a systémových zdrojů [4]. Kontejnerové služby najdou využití při

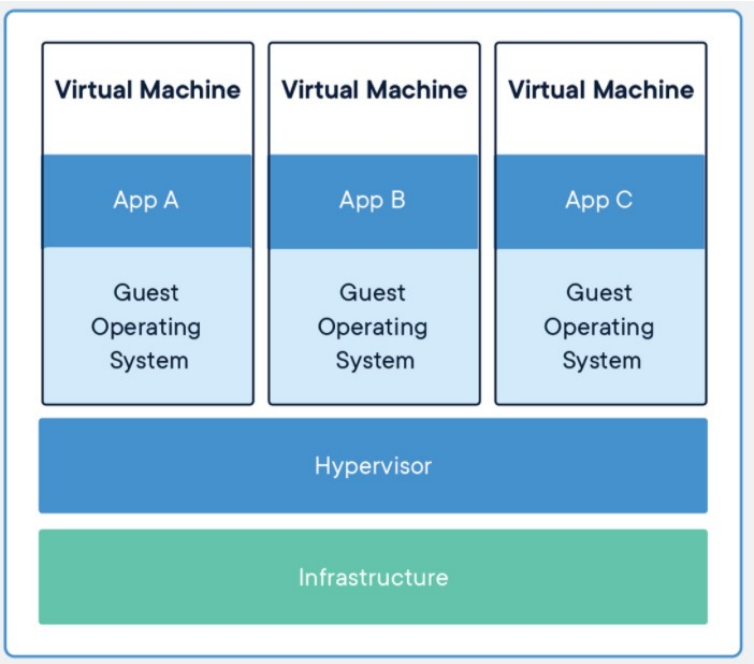


Obrázek 2.1: Kontejnerizace [1]

řešení diplomové práce v oblasti migrace. Jak již bylo dříve uvedeno, takto vytvořený kontejner lze jednoduše spustit v jakémkoliv výpočetním prostředí nehledě na platformu či hardware.

2.2 Virtualizace

V předešlé podkapitole jsme se zaobírali kontejnerizací, což je virtualizace operačního systému. Nyní si popíšeme, co je virtualizace obecně, zejména virtuálního stroje. Pomocí virtualizace vytváříme virtuální výpočetní prostředí namísto fyzického, to zahrnuje počítačem vytvořené verze hardware a operačních systémů. Díky tomu můžeme rozdělit jeden fyzický počítač na několik virtuálních, kdy každý takový počítač pracuje nezávisle a může na něm běžet jiný operační systém a aplikace [5]. Mezi zástupce virtualizačního software patří dobře známý a hodně využívaný VirtualBox, který se řadí mezi nekomerční programy. Dále je to například VMware, Qemu nebo Bochs. Mezi komerční software patří program od firmy Microsoft Hyper-V [6]. Virtualizace má mnoho využití, jedním z příkladů může být jednoduchá škálovatelnost. Dle vlastních potřeb můžeme virtuálnímu stroji přidávat nebo ubírat prostředky, které využívá a tím řídit jeho výkon podle aktuální potřeby [7]. Dalšími případy využití jsou zálohování a migrace. Celý virtuální stroj je tvořen několika soubory, migrace takového počítače je tedy pouze otázkou přenosu těchto souborů na jiný fyzický počítač s virtualizačním programem. Totéž platí i pro zálohování. Pro kompletní zálohu virtuálního stroje se všemi aplikacemi a soubory stačí pouze vytvořit kopii těchto souborů [8]. Stejně jako použití



Obrázek 2.2: Virtualizace [1]

kontejnerů může mít i virtualizace celého počítače využití při řešení diplomové práce. Příkladem může být třeba zálohování informačního systému, který by běžel na virtuálním stroji, jeho migraci na jiný fyzický počítač nebo škálování výkonu podle měnící se zátěže.

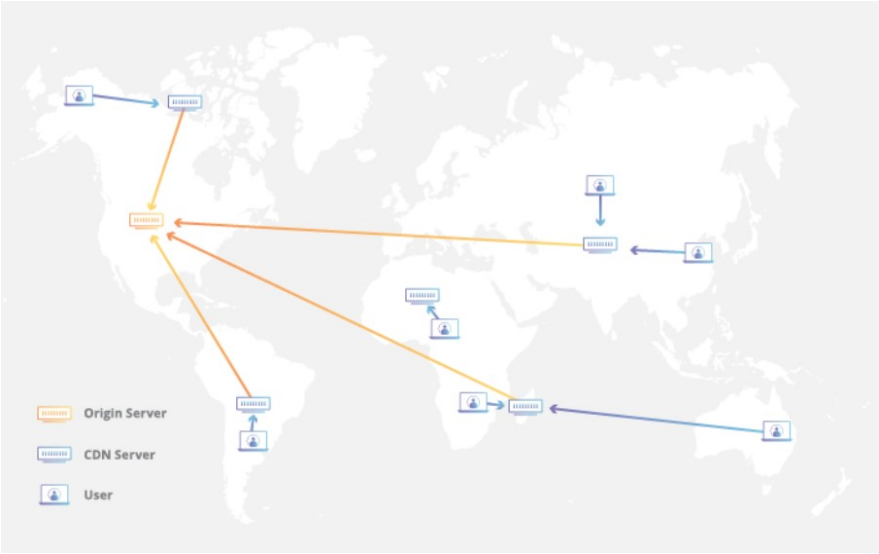
2.3 Cloud

Nyní se přesuneme od virtualizace různých úrovní počítačového systému k využití technologie počítačové sítě, cloudu. Jedná se o aplikace a služby, které jsou uživatelům dostupné přes internet na serverech jejich poskytovatele. Tyto servery tvoří globální síť, jsou vzájemně propojené, komunikují spolu a fungují jako jeden ekosystém [9]. Mezi možné nabízené služby patří například ukládání a správa souborů (Google drive), streamování videí, webová pošta nebo kancelářské aplikace (dokumenty, tabulky, prezentace...). U cloudu můžeme rozlišit tři typy služeb. První je tzv. software jako služba (SaaS), která znamená, že poskytovatel cloudu hostuje zákazníkovu aplikaci a ten nemusí řešit a udržovat vlastní infrastrukturu. Dalším typem je platforma jako služba (PaaS). Tento model poskytuje přístup k vývojářským nástrojům pro vytváření a správu aplikací [10]. Posledním typem je infrastruktura jako služba (IaaS). Zde provozovatel cloudu poskytuje zákazníkovi infrastrukturu. Typicky se jedná o virtualizaci [11]. Mezi nejznámější a nejpoužívanější poskytovatele cloudových platforem patří Microsoft Azure, Amazon web services, Google cloud services a Oracle cloud infrastructure. Jednou z hlavních výhod použití cloudu je jeho vysoká škálovatelnost. Jak již z definice vyplývá, jedná se o spousty serverů, které spolu komunikují. Můžeme tedy dle naší potřeby přidávat

nové prostředky a tím jednoduše navýšit výkon a kapacitu. Dalšími výhodami, které můžeme zmínit je vysoká dostupnost (ke cloudu se můžeme připojit z jakéhokoliv zařízení s přístupem na internet), zabezpečení dat [12]. Při řešení diplomové práce můžeme využít služeb některé z cloudových platforem pro snadnou škálovatelnost webové aplikace, nebo například pro migraci databázových serverů pomocí platformy Microsoft Azure.

2.4 Content delivery network

Síť pro doručování obsahu, zkráceně CDN, je další technologií v oblasti počítačových sítí, která může být využita při vytváření řešení diplomové práce. Budeme se ji tedy v této podkapitole věnovat. Z názvu této technologie jednoznačně vyplývá, že cílem je doručování internetového obsahu koncovým uživatelům, a to co možná nejrychleji. Tímto obsahem může být cokoliv, od obrázků a videí až po obyčejné webové stránky a soubory javascriptu. Služby CDN jsou každým dnem populárnější a dnes již většina webového provozu na stránkach Facebook, Twitter nebo Amazon je obsluhována touto sítí [13]. Síť pro doručování obsahu se skládá z geograficky distribuovaných, vzájemně propojených serverů skrze internet. CDN se skládá z několika částí. První z nich je počáteční server, kde dochází k distribuci obsahu. Další částí jsou servery rozmístěné po celém světě. Na tyto servery se replikuje obsah počátečního serveru. Jako poslední obsahuje směrovací systém, který slouží k doručení obsahu k uživateli z geograficky nejbližšího uzlu [14]. Na výběr máme spoustu poskytovatelů služby CDN. Většina z nich nabízí zkušební verzi, která se liší podle vybraného poskytovatele (časové omezení, omezení využitých GB...). Službu doručování obsahu nabízí například Microsoft Azure CDN, CloudFlare, Amazon CloudFront nebo CDN77.



Obrázek 2.3: Content delivery network [13]

Síť doručování obsahu se používá zejména pro zrychlení načítání webových stránek. Toto zrychlení je dosaženo tím, že požadavek na internetový obsah bude zpracován zeměpisně nejbližším CDN serverem místo počátečním serverem, který může být od uživatele stovky kilometrů daleko. Dalším benefitem této technologie je vysoká dostupnost obsahu. Díky existenci desítek, stovek případně až tisíců uzlů v síti, výpadek jednoho či více serverů nemusí znamenat i výpadek celé webové aplikace. V rámci řešení závěrečné práce se síť doručování obsahu dá využít pro zálohování multimediálního obsahu, jelikož dochází k replikaci dat z hlavní uzlu na ostatní servery. Dále můžeme tuto technologii využít pro navyšování výkonu webových aplikací. CDN je velmi dobře škálovatelná, můžeme tedy výkon libovolně navýšit nebo snížit podle měnící se zátěže webu.

Shrnutí

Z předchozích podkapitol můžeme o zmíněných technologiích shrnout to nejdůležitější, co jsme se z nalezených zdrojů informací o nich dozvěděli.

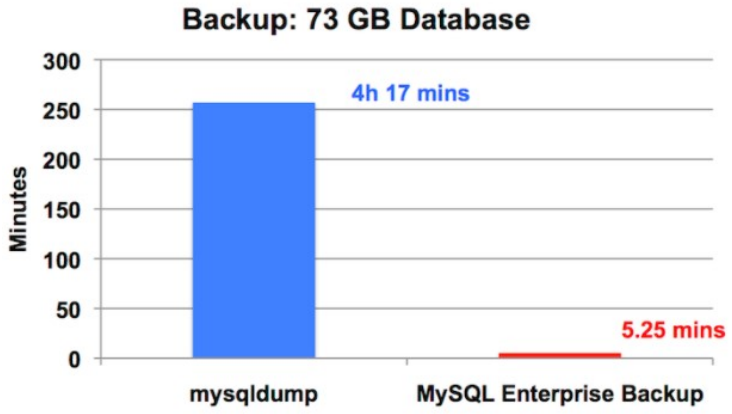
- **Kontejnerové služby** - kontejner je standardizovaná jednotka software, virtualizace operačního systému, vhodné zejména pro migraci aplikací, testování
- **Virtualizace** - virtuální výpočetní prostředí (hardware + operační systém), rozdělení jednoho fyzického počítače na více virtuálních, dobrá škálovatelnost virtuálních strojů, zálohování a jejich migrace
- **Cloud** - aplikace a služby na serverech poskytovatele, globální síť vzájemně komunikujících serverů, dobrá škálovatelnost webových aplikací, migrace databázových serverů
- **Content delivery network** - geograficky distribuované a vzájemně propojené servery, distribuce internetového obsahu (obrázky, videa, webové stránky...), dobrá škálovatelnost webových aplikací, zálohování obsahu

Nyní tedy víme, jaké technologie můžeme pro řešení zadaných problémů použít a jak fungují. V následujících kapitolách se tedy můžeme přesunout k návrhu a implementaci jednotlivých částí diplomové práce a zmíněné přístupy správně použít.

Kapitola 3

Edice databázových systémů

Drtivá většina vývojářů open-source databázových systémů vydává více než jednu verzi svého produktu. Jednou z nich je základní edice, kterou poznáme podle názvu "community", "standard" nebo "basic". Tyto edice jsou, co se týče dostupných funkcí, velmi strohé oproti vyšším enterprise edicím. Dalším nedostatkem základních verzí databází může být jejich výkon. Slabší výkon se může projevit například u zálohování, kdy placená enterprise edice poskytuje více možností zálohování dat a rychlejší vykonání. Jako příklad můžeme uvést oficiální porovnání výkonu zálohování pomocí nástroje mysqldump ve verzi community a zálohovacího nástroje pro enterprise edici databázového systému MySql v následujícím grafu.



Obrázek 3.1: Mysqldump vs Enterprise backup [15]

V následující podkapitole se podíváme na další funkce v MySQL Enterprise edition a porovnáme ji s prémiovými verzemi ostatních databázových systémů.

3.1 MySQL Enterprise Edition

Nejdůležitější součástí enterprise edice databázového systému MySQL je nástroj pro vytváření zálohování a obnovy databází. Díky němu můžeme provést různé metody zálohování a obnovy z příkazové řádky a několikanásobně navýšit výkon těchto operací. Jak už jsme vypozorovali z obrázku v předchozí kapitole, tak v případě zálohování se jedná až o 49x lepší výkon, u operace obnovy je toto zlepšení až 80ti násobné. V nástroji mysqlbackup můžeme provést následující metody zálohování:

- **InnoDB Hot backup** - jedná se o metodu zálohování běžící MySQL instance (opak tzv. Cold backup) s minimálním narušením volaných operací na datech. [15]

- **Compressed backup** - pomocí kompresní metody ušetříme místo na disku a také zvýšíme rychlost zálohování. Kompresi můžeme použít pouze pro InnoDB tabulky. [15]

- **Partial backup** - tento způsob použijeme, pokud chceme zálohovat pouze vybraná data. Výchozím nastavením je záloha všech souborů, které se nacházejí v datové složce databáze. Můžeme například zahrnout nebo vyloučit tabulky pomocí *–include-tables* a *–exclude-tables*. [15]

- **Incremental backup** - je vhodná použít pro data, ve kterých nedochází k velmi častým a rozsáhlým změnám. Jako první se provede úplná záloha databáze a poté se provádějí inkrementální zálohy, které obsahují pouze změny provedené od posledního přírůstku. Nevýhodou této metody je delší čas obnovy databáze. [15]

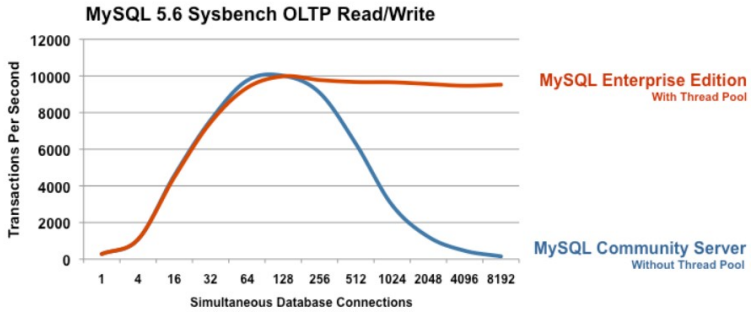
- **Full backup** - úplná záloha je základem pro většinu ostatních metod (např. inkrementální) a vytvoří nám celkovou zálohu MySQL instance (všechny databáze a tabulky). [15]

- **Optimistic backup** - jedná se o funkci, která zvyšuje výkon zálohování a obnovy velkých databází, ve kterých je malý počet často modifikovaných tabulek. Během hot backup velkých databází mohou vznikat obrovské redo log soubory a ty se mohou zvětšovat rychleji, než je nástroj mysqlbackup může zpracovat. To může skončit selháním operace zálohování. Řešením je optimistic backup, který má dvě fáze. V tzv. optimistické fázi jsou tabulky, které nebudou během zálohy modifikovány, označeny jako neaktivní. Ty jsou zálohovány bez jakýchkoliv zámků a bez redo a undo log souborů. Ve druhé fázi dochází k běžné záloze tabulek, redo a undo logů, které nebyly zálohovány v první fázi. Tento optimistický přístup velmi výrazně snižuje dobu potřebnou k zálohování a obnově databáze. [15]

Ke všem těmto metodám zálohování existují funkce, které slouží pro obnovení databáze z daného typu provedené zálohy. Enterprise edice MySQL nám také nabízí nové možnosti zabezpečení. Tyto možnosti jsou následující:

- **Authentication** - pomocí této enterprise funkce můžeme použít pro autentizaci uživatelů do databázi v MySQL instanci již existující infrastrukturu zabezpečení jako je například Linux Pluggable Authentication modules nebo Windows active directory. [15]
- **TDE** - funkce transparent data encryption slouží k data-at-rest šifrování citlivých dat a informací, zabránění jejich úniku, ochraně soukromí a splnění regulačních požadavků jako jsou standardy o ochraně dat kreditních karet. Fyzické soubory databáze se šifrují automaticky před jejich zápisem do databáze a dešifrují se během jejich čtení. [15]
- **Encryption** - pro zabezpečení citlivých dat poskytuje MySQL Enterprise edice funkcionalitu asymetrického šifrování. K tomu patří šifrování a dešifrování, generování klíčů, digitální podpisy a další. [15]
- **Masking** - další funkce z oblasti zabezpečení MySQL slouží k ochraně citlivých údajů před neautorizovanými uživateli pomocí skrývání skutečných informací a jejich nahrazení falešnými daty. Díky tomu můžeme zajistit dodržení zákonů o ochraně osobních údajů jako je GDPR a zamezíme možnému úniku dat. [15]
- **Firewall** - firewall nás chrání proti kyberútokům specifickými pro databázové systémy. Naše data chrání pomocí monitorování, upozornění a blokováním neautorizované databázové aktivity. [15]
- **Audit** - posledním nástrojem z oblasti bezpečnosti je systém pro logování přístupu k jednotlivým datům, pokusy o přihlášení a odhlášení, změny databázového schématu, přístup k databázi, tabulkám a další. [15]

Další skupinou nových funkcionalit, ke kterým máme v enterprise edici přístup, jsou nové nástroje a funkce pro škálovatelnost databázového systému. Ta je ve světě databází velmi důležitou součástí pro zajištění dobrého výkonu rozsáhlých systémů. V MySQL je základním nástrojem pro škálovatelnost replikace a enterprise edice přidává možnost tzv. Thread Pool. Ten nám poskytuje lepší zpracování jednotlivých vláken a tím snižuje režii spravování jednotlivých uživatelských připojení a vláken vykonávajících dotazy. Díky tomu získáme až 60 krát lepší škálovatelnost [15].



Obrázek 3.2: Sysbench OLTP Read/Write [15]

Poslední funkcionalitou enterprise edice je poskytnutí vysoké dostupnosti databází. Tento pojem nám říká, že naše data budou dostupná, pokud možno, nepřetržitě i v případě možných poruch a selhání. To je zajištěno tzv. skupinovou replikací (group replication). Ta nám poskytuje například elastičnost, která je dosažena dynamickou změnou členů skupiny a její následnou rekonfigurací. Dále je to detekce selhání, kdy server, na kterém došlo k chybě, je automaticky vyřazen ze skupiny a jako poslední je zajištění odolnosti proti chybám. [15].

3.2 MongoDB Enterprise

Stejně jako MySQL má i databázový systém MongoDB enterprise edici, která nabízí nové funkce pro kvalitnější a bezpečnější provoz našich databází. První skupinou enterprise funkcí je správa a řízení. Zde je dostupný nástroj Ops manager, který nám nabízí následující: monitoring a vizualizaci více než sto výkonnostních metrik s možností nastavení upozornění, když se jedna z metrik dostane mimo rozsah přijatelných hodnot, automatizaci administrativních úloh a zálohování replika setů pomocí point-in-time zálohy a clusterů použitím snapshotů [16]. Dále jsou zde dostupné možnosti ochrany dat, zabezpečení a řízení přístupu pomocí funkcí pokročilého zabezpečení. Ty nám umožňují například ověření uživatelů pomocí Kerberos a LDAP, auditování, šifrování dat v perzistentním uložišti (tzv. data at rest) použitím AES256-CBC a TLS/SSL šifrování komunikace [16]. Kromě těchto dvou zmíněných sad funkcí nabízí enterprise edice dále například integraci nástroje Kubernetes pro orchestraci kontejnerů, podpora SNMP, podpora in-memory storage engine, který nabízí mnohem rychlejší databázové operace a konektor, pomocí kterého bude možné použít MongoDB databázi pro business intelligence nástroje, které pracují s relačními daty [16].

3.3 Elasticsearch

Fulltextový vyhledávač elasticsearch nabízí řadu různých edic, které postupně nabízejí více funkcí a nástrojů pro správu, monitoring a zabezpečení elasticsearch clusterů. Těmito edicemi jsou standard, gold, platinum a enterprise. Abychom mohli v následující podkapitole porovnat jednotlivé edice všech zmíněných databázových systémů, zaměříme se pouze na enterprise verzi, která nabízí veškeré funkce a nástroje dostupné v systému Elasticsearch. Z hlediska zabezpečení je podporována řada stejných služeb jako v případě předchozích databází. Jedná se zejména o podporu šifrování dat v perzistentním úložišti, šifrování komunikace pomocí TLS/SSL, řízení přístupu pomocí rolí, ověření uživatelů přes LDAP a Kerberos, auditování, IP filtrování a nástroj SIEM pro detekci hrozeb a Elastic Agent pro prevenci malware [17]. Dále máme možnost využít strojového učení pro detekci anomálií v datech nebo pro identifikaci jazyka. Za zmínku také stojí nástroje a vlastnosti pro automatické horizontální škálování, clusterování a shardování [18]. V poslední řadě existují nástroje pro monitorování, vizualizaci a logování různých metrik.

3.4 Porovnání

Nyní když už známe jednotlivé enterprise edice námi zkoumaných databázových systémů, můžeme se pustit do porovnání mezi jednotlivými systémy a mezi standardními a enterprise edicemi. Začněme tedy porovnáním bezplatné základní edice se zpoplatněnou rozšířenější verzí. Zřejmě nejpodstatnějším rozdílem je úroveň zabezpečení, kterou má verze enterprise rozšířenou o spoustu nových funkcí a nástrojů. Díky tomu jsou veškerá data velmi dobře zabezpečená před zneužitím, ztrátou a odcizením. Dalším jednoznačným rozdílem je výkon jednotlivých databází v rozšířených edicích. Jako příklad můžeme uvést rychlost zálohy a obnovy v MySQL enterprise oproti MySQL community server (viz 3.1) nebo zvýšená rychlost databázových operací při použití in-memory storage engine v MongoDB. Posledním rozdílem, který můžeme zmínit, je nabídka různých nástrojů pro monitorování a vizualizaci různých metrik databázového systému. Můžeme tedy říct, že enterprise edice jsou velmi důležité pro práci s rozsáhlými databázemi, které obsahují citlivá data a jsou nezbytnou součástí informačních systémů. V následující tabulce je uveden sumář některých funkcí z rozšířených verzí databázových systémů MySQL, MongoDB a Elasticsearch.

Tabulka 3.1: Porovnání funkcí edic databázových systémů

Funkce	MySQL Enterprise	MongoDB Enterprise	Elasticsearch
Autentizace	✓	✓	✓
Šifrování (data at rest)	✓	✓	✓
Audit	✓	✓	✓
Škálovatelnost	✓	✓	✓
TLS/SSL	X	✓	✓
Pokročilé zálohování a obnova	✓	✓	✓
Monitoring a vizualizace	✓	✓	✓
Firewall, Anti-malware	✓	X	✓
Machine learning	X	X	✓

Můžeme vidět, že enterprise edice vybraných systému obsahují stejné či velmi podobné funkce a nástroje, které zajišťují bezpečný chod, dobrý výkon a kvalitní správu databázového systému a jsou základem pro každou rozšířenou verzi.

Kapitola 4

Zálohování a replikace

V první části diplomové práce se budeme věnovat problematice zálohování databázových systémů. Nejprve si představíme existující software, který nám nabízí možnosti rychlého a jednoduchého zálohování dat. Následně se podíváme na způsoby zálohování, zejména na datová úložiště, na které můžeme ukládat data, typy záloh a zálohování s nepřetržitým provozem pomocí master-slave replikace. Dále si ukážeme, jak se provádějí zálohy a jak se nastavuje replikace v databázových systémech MySQL, MongoDB a Elasticsearch. V poslední části kapitoly se budeme věnovat výsledkům testování záloh v jednotlivých databázích.

4.1 Existující software

Pro zálohování databází MySQL existuje mnoho programů, pomocí kterých můžeme snadno a lehce zálohovat naše uložená data. Kromě různých možností zálohy samozřejmě také poskytují funkce pro obnovu databáze.

Nástroj Iperius backup [19] slouží k vytváření záloh různých databázových systémů (MySQL, MariaDB, Oracle, PostgreSQL, Microsoft SQL Server). Mezi jeho klíčové charakteristiky patří například export tabulek, pohledů, uložených procedur, plánování automatických záloh a jejich pokročilá verifikace. Dále máme možnost vytvořené soubory komprimovat a šifrovat pomocí AES a ukládat je na různá úložiště včetně cloudu (například Google drive). Aplikace je dostupný za jednorázový poplatek 149 eur.

SQLBackupAndFTP [20] je nástroj, který obsahuje velmi podobné funkce jako Iperius backup, ale obsahuje i pár novinek. Nejpodstatnější novinkou je systém notifikací, který nás informuje o stavu provedených záloh, tedy jestli námi naplánovaná záloha proběhla úspěšně nebo zdali došlo k jejímu selhání. Další novou vlastností je možnost prohlížení historie o všech provedených zálohách online. Jedná se opět o komerční software, který je dostupný v několika edicích. Některé funkce nástroje jsou dostupné pouze ve vyšších edicích. Jejich cena se pohybuje od 0$ až po 499$.

Jedním z nejrozšířenějších nástrojů pro zálohování a obnovu dat je Percona XtraBackup [21]. Jedná se o open source aplikace, která je dostupná zdarma a poskytuje rychlé a spolehlivé zálohy (hot backup, inkrementální zálohy atd.). Stejně jako předchozí nástroje tak i Percona XtraBackup nabízí možnosti automatizace zálohování, kompresy dat a verifikaci zálohy.

4.2 Způsoby zálohování

Nejdůležitější částí každého informačního systému jsou data, se kterými uživatelé pracují. Těmi mohou být například informace o produktech, osobní údaje lidí, faktury a další. Jedním slovem se jedná o velice důležité business údaje, jejichž ztrátě (ať už z důvodu softwarové nebo hardwarové chyby nebo např. hackerského útoku) musíme zamezit. K tomu nám slouží zálohování, což je jedna z nejdůležitějších praktik správy databázových systémů. Jedná se o vytvoření kopie dat, která jsou uložena v libovolném databázovém systémů, na jiný datový nosič (USB flash disk,optický disk, externí SSD/HDD disk, magnetická páska, cloud atd).

Existuje několik typů záloh. Nejzákladnějším typem je úplná záloha, při které dochází k zálohování celého databázového systému (případně pouze specifikovaných databází, tabulek). Dále můžeme provést zálohu například inkrementální metodou (viz podkapitola 4.6) nebo rozdílovou metodou. Pro zálohování dat je důležité vybrat správnou metodu zálohování. Úplná záloha není vhodná pro rozsáhlé databáze, protože vytváření takové zálohy trvá dlouhou dobu a zabírá spoustu místa v úložném prostoru. Proto je vhodnější použít inkrementální zálohování, které tyto problémy řeší. Je dobrou praktikou v databázovém systému nastavit automatické provádění záloh ve vhodně zvoleném časovém intervalu. Tento interval by se měl odvíjet podle četnosti změn a objemu změněných dat. U inkrementálních záloh se typicky jedná o úplnou zálohu jednou týdně se zálohou přírůstků jednou každý den. Zálohy mohou být prováděny v režimu offline (databázový server se nenachází v běžném chodu) nebo online (běžný chod databáze). Samozřejmostí je vytváření kopií dat za běžného chodu, aby mohli být služby informačního systému dostupné nepřetržitě. Zálohovací proces ale může být velmi náročný na výkon databázového systému a veškeré operace prováděné uživateli by byly zpracovány s velkým zpožděním. To je samozřejmě nežádoucí efekt, kterému musíme předcházet. Toho dosáhneme využitím master-slave replikace. Proces zálohování poté vypadá následovně: jeden ze serverů běží (master) v běžném režimu a na druhém serveru (slave) dochází k zálohování dat a k případné údržbě databáze (kontrola integrity databáze, defragmentace indexů, znovusestavení indexů, údržba log souborů...). Následně dojde k synchronizaci a tyto kroky se poté provedou pro master server. Tímto způsobem tedy můžeme provést zálohování databázových systému za nepřetržitého provozu bez omezení výkonu a funkčnosti hlavního serveru, na kterém probíhá provoz. V následujících podkapitolách se podíváme na implementaci master-slave replikace a zálohování na třech vybraných systémech.

4.3 Zálohování a replikace v MySQL

Relační databázový systém MySQL nám nabízí spoustu možností, jak zálohovat naše data. Jak již bylo dříve zmíněno, jde nám o zálohování s nepřetržitým provozem a z tohoto důvodu je třeba nejdříve nastavit replikaci. K tomu máme k dispozici dva virtuální stroje, na kterých běží instance MySQL. Pro nastavení replikace musíme provést úpravu konfiguračního souboru my.cnf pro oba servery. Na serveru, který bude sloužit jako master v naší replikaci nastavíme proměnnou server-id na hodnotu 1, cestu k binárním logům a jména databází, které se nebudou replikovat. Podobné nastavení provedeme i na slave serveru s jediným rozdílem, hodnota server-id bude rovna 2. Dalším krokem je vytvoření uživatelského účtu, který bude provádět replikaci a následně mu udělit replikační práva. To provedeme následujícím příkazem na master serveru.

```
GRANT REPLICATION SLAVE ON *.* TO 'replika_ucet'@'adresa_slave' IDENTIFIED BY '
    heslo';
```

Listing 4.1: Přidání práv pro replikaci

Posledním krokem je provedení následujících příkazů na slave serveru, čímž spustíme replikaci.

```
CHANGE MASTER TO MASTER_HOST='adresa_master', MASTER_PORT=3306, MASTER_USER='
    replika_ucet',MASTER_PASSWORD='heslo', MASTER_LOG_FILE='mysql-bin.000001',
    MASTER_LOG_POS=1;

START SLAVE;

SHOW SLAVE STATUS \G
```

Listing 4.2: Spuštění replikace

Poslední příkaz nám zobrazí výpis stavu replikace a případné chyby, které během běhu nastaly. Nyní se můžeme přesunout k vytváření záloh dat, která se nacházejí v naší databázi. První způsob, který můžeme použít je vytvoření úplné zálohy pomocí programu *mysqldump*. Výsledkem je .sql soubor, který obsahuje dotazy CREATE pro vytvoření objektů (tabulky, databáze) a dotazy INSERT pro vložení dat. Další možností, jak zálohovat, je kopírování souborů databázových tabulek. U této metody musíme zajistit uzamčení tabulek zámkem pouze pro čtení a zapsání veškerých změn z cache paměti na disk. Poté už stačí pouze zkopírovat dané soubory do cílového úložiště. Dále pak můžeme využít možnosti exportu dat například do souboru ve formátu .csv pomocí dotazu SELECT * INTO OUTFILE nebo vytvoření snapshotu souborového systému. Poslední možností je inkrementální záloha, při které kopírujeme binární log soubory.

4.4 Zálohování a replikace v MongoDB

V dokumentové NoSQL databázi MongoDB mluvíme o replikaci jako o "replica set". Její nastavení je velice jednoduché. Na master a slave serverech musíme provést následující úpravy v konfiguračním souboru *mongod.conf*.

```
replication:
   replSetName: "unikatni nazev"
net:
   bindIp: localhost,<hostname(s)|ip address(es)>
```

Listing 4.3: Nastavení konfiguračního souboru MongoDB

Poté co provede nutné změny v konfiguraci MongoDB instancí, které chceme zařadit do naší množiny replik, spustíme shell na master serveru. Zde zadáme následující příkaz pro nastavení a spuštění replikace.

```
rs.initiate( {
   _id : "rs0",
   members: [
      { _id: 0, host: "<IP adresa>:27017" },
      { _id: 1, host: "<IP adresa>:27017" }
   ]
})
```

Listing 4.4: Spuštění replikace v MongoDB

Správnost konfigurace můžeme zkontrolovat příkazem rs.conf(). Poté co máme funkční replikaci a naimportovaná data, můžeme provést zálohování. K tomu můžeme použít například program *mongodump*, jehož výstupem je soubor ve formátu .bson a obsahuje všechny dokumenty, které se nacházejí v zálohované databázi nebo kolekci. Jedná se tedy o úplnou zálohu. Pokud během operace mongodump dochází k ukládání nových dokumentů nebo aktualizačních operací, tak je nutné použít parametr –oplog, který vytvoří soubor oplog.bson, který obsahuje změny provedené během zálohování. Jedná se tedy o vytvoření přírůstku a provedení inkrementální zálohy [22]. Další možností, jak můžeme provést zálohování, je vytvoření snapshotu souborového systému na platformě Linux pomocí nástroje LVM. Dále můžeme zálohovat pomocí nástroje *mongoexport* do souborů ve formátu .csv a .json nebo přímým zkopírováním datových souborů. Ve druhém případě musíme zajistit, že nedochází k žádným zápisům dat, jinak mohou být zkopírované soubory v neplatném stavu [22].

4.5 Zálohování a replikace v Elasticsearch

V tomto databázovém systému je prvním krokem vytvoření elasticsearch clusteru. Stejně jako v případě předchozích databází musíme provést úpravy v konfiguračním souboru. Tento soubor se jmenuje elasticsearch.yml a v něm nastavíme jméno clusteru, uzlu, adresy serverů a jméno master uzlu. Výsledná konfigurace může vypadat následovně.

```
cluster.name: diplomka
node.name: serverA
network.host: 0.0.0.0
discovery.seed_hosts: ["158.196.145.77", "158.196.145.88"]
cluster.initial_master_nodes: ["serverA",]
```

Listing 4.5: Konfigurace elasticsearch.yml

Podobné nastavení se provede také na slave serveru, pouze s tím rozdílem, že hodnota node.name bude mít jinou hodnotu (např. serverB). Po provedení těchto změn a restartování elasticsearch služby můžeme zkontrolovat náš cluster následujícím HTTP požadavkem.

```
curl -XGET http://158.196.145.77:9200/_cat/nodes?v
```

Listing 4.6: Kontrola nastaveného clusteru

Po nastavení funkčního elasticsearch clusteru můžeme přejít k vytváření záloh. V systému elasticsearch existuje jediná možnost, a to je vytváření snapshotů. Jedná se o zálohu běžícího elasticsearch clusteru, která může obsahovat veškerá data a indexy, které se v clusteru nacházejí, nebo pouze specifikované indexy [23]. Před vytvořením snapshotu musíme nejprve vytvořit repozitář následujícím požadavkem.

```
PUT /_snapshot/elastic_backups
{
  "type": "fs",
  "settings": {
    "location": "/data/elastic_snapshots/elastic_backups"
  }
}
```

Listing 4.7: Vytvoření repozitáře pro snapshot

Tento repozitář se může nacházet jak na lokálním stroji, tak i na vzdálených úložištích jako je například Amazon S3, Microsoft Azure nebo Google Cloud Storage [23]. Jakmile je umístění repozitáře nastaveno, můžeme přejít k vytvoření snapshotu.

```
PUT /_snapshot/elastic_backups/snapshot_1?wait_for_completion=true
```

```
{
  "indices": "produkty",
  "ignore_unavailable": true,
  "include_global_state": false
}
```

Listing 4.8: Vytvoření snapshotu

Vytváření těchto snapshotů je inkrementální zálohování. Každý vytvořený snapshot obsahuje pouze data, která nejsou obsažena v tom předešlém. Díky tomu můžeme zálohovat velmi často s minimálním zatížením serveru a malou velikostí záloh [23].

4.6 Inkrementální zálohování

Jedná se o jedny z možných metod, které můžeme použít pro zálohování našich dat. Fungování těchto metod je celkem jednoduché. Nejprve dojde k vytvoření celkové zálohy databázového systému a poté dochází k vytváření kopií dat, která se změnila od poslední celkové zálohy. Jedná se o velmi oblíbené a často využívané techniky zálohování dat, což plyne z jejich výhod. Jelikož se jedná pouze o nová či změněná data od poslední provedené zálohy, zabírají méně místa na disku a jejich vytvoření je rychlé. Metoda inkrementálních záloh se hodí pro rozsáhlé databáze, jejichž celková záloha by se vytvářela dlouhou dobu a zabírala by spoustu místa v úložném prostoru. Jejich nevýhodou je zvýšená doba obnovy dat, jelikož musíme nejprve obnovit celkovou zálohu databáze a poté všechny soubory obsahující přírůstky. Existuje několik typů inkrementálních záloh, které můžeme použít, ty jsou následující:

- **File-level** - pokud dojde ke změně v souboru, tak je přesunut do repozitáře a vytvoří se jeho nová verze. Vhodné pro malá data.
- **Block-level** - dochází k záloze pouze změněných částí (bloků) místo celých dat.
- **Byte-level** - monitorování a záloha jednotlivých bytů, ve kterých došlo ke změně.
- **Reverse** - k úplné záloze se připojí nově vytvořený přírůstek a vytvoří se nová úplná záloha. Tento typ nám poskytuje efektivnější obnovu, jelikož se provádí z jednoho souboru místo z několika přírůstků.
- **Multilevel** - schéma, které obsahuje několik úrovní záloh. Úroveň n zálohuje veškeré změny, které nastaly od poslední zálohy úrovně n-1.

4.7 Výkonnostní testování

Ve zvolených databázových systémech MySQL, MongoDB a Elasticsearch byly provedeny výkonnostní testy na 4 milionech záznamů, pomocí kterých se zjistila rychlost zálohy a obnovy během běžící zátěže. Ta je vytvořena v jazyce Python a na databázi volá v náhodně zvoleném počtu vláken (3 až 7) select dotazy. Výsledky měření jsou následující.

Tabulka 4.1: Výsledky výkonnostního testování pro úplnou zálohu

Databáze	Úplná záloha (bez vytížení)	Úplná záloha (s vytížením)	Obnova DB
MySQL	725s	910s	2733s
MongoDB	261s	269s	788s
Elasticsearch	196s	205s	272s

Tabulka 4.2: Výsledky výkonnostního testování pro inkrementální zálohu

Databáze	Inkrementální záloha	Obnova DB
MySQL	21s	2705s
MongoDB	107s	1479s
Elasticsearch	166s	-

Z tabulek výsledků výkonnostního měření (tabulky 4.1 a 4.2) můžeme vypozorovat, že databázový systém MySQL je ze tří vybraných databází z hlediska vytváření záloh a obnovy nejpomalejší. Také má na něj velký vliv běžící vytížení, které vytvoření úplné zálohy zpomalilo o 185 sekund. Inkrementální zálohování je ale u MySQL velmi rychlé, jelikož se jedná o zálohování (kopírování) binárních log souborů, které vznikají při jakékoliv manipulaci s daty (vkládání, mazání, aktualizace). Systém Elasticsearch je z hlediska rychlosti vytváření záloh a obnovy nejrychlejší a běžící zatížení stejně jako u MongoDB nemělo na rychlost zálohování moc velký vliv. U Elasticsearch nám chybí v tabulce záznam pro obnovu z inkrementální zálohy. Je to z důvodu, který vyplývá ze samotného vytváření záloh (tzv. snapshotů) v této databázi (viz podkapitola 4.5). Proto se pro obnovu indexu použije pouze naposledy vytvořený snapshot místo několika vytvořených souborů jako v případě MongoDB a MySQL.

Shrnutí

Ukázali jsme si tři nástroje, které můžeme použít pro zálohování, jedná se o komerční Iperius Backup a SQLBackupAndFTP a open-source Percona XtraBackup. Dále jsme se zmínili o důležitosti

zálohování databází, používaných datových nosičích pro uložení dat (SSD, HDD, magnetická páska, optický disk, cloud atd.), různých typech záloh (úplná, inkrementální, rozdílová metoda) a zálohování s nepřetržitým provozem pomocí master-slave replikace. Z hlediska zálohování vybraných databázových systémů jsme si představili například nástroj mysqldump pro systém MySQL, mongoexport pro databázi MongoDB a vytváření snapshotů v Elasticsearch. Na konci kapitoly jsme provedli srovnání rychlostí úplné a inkrementální zálohy v databázích MySQL, MongoDB a Elasticsearch.

Kapitola 5

Migrace databázových systémů

Tato kapitola je věnována problematice migrace databázových systémů. Po představení několika existujících nástrojů pro migraci databází se budeme věnovat definici samotného migračního procesu, tj. co si pod termínem migrace máme představit. Následně se dozvíme o možných případech využití tohoto procesu a cílových systémech. Dále si popíšeme postup migrace databázového systému a možné problémy, které musíme během této aktivity vyřešit. Na konci této kapitoly bude zmíněno několik benefitů databázové migrace.

5.1 Existující řešení

Pro databázovou migraci existuje spousta nástrojů. Některé jsou jednoduché a provádějí databázovou migraci pouze mezi databázemi stejného typu (např. mezi relačními databázemi) a některé jsou sofistikovanější a nabízejí migraci například mezi relační a dokumentovou databází, kde je potřeba provést složitou konverzi databázového schématu. Většina nástrojů se od sebe navzájem moc neliší a poskytují uživatelům velmi podobné funkce. Jedinými odlišnostmi jsou tak jen podporované databáze a uživatelské rozhraní. Na závěr si můžeme pár nástrojů vyjmenovat:

- **DBConvert Studio** - komerční software, podpora pouze relačních databází (SQL server, MySQL, Oracle, Firebird, MariaDB a další), podpora migrace do cloudů
- **SQL Data Examiner** - komerční nástroj, opět podpora pouze relačních databází
- **Altova MapForce** - licencovaný program, který dokáže migrovat mezi různými typy databází a provádět konverze schémat (SQL, JSON, XML)
- **Flyway** - open-source migrační nástroj pro relační databáze s možností migrace do cloudu
- **Dbmate** - open-source program, který migruje MySQL, PostgreSQL a SQLite

5.2 Definice migrace

Databázová migrace se dá definovat jako proces, během kterého dochází k migraci, jiným slovem k přemístění, dat ze zdrojové (anglicky source) databáze či vícero databází do cílové (anglicky target) databáze nebo databází. Po tomto migračním procesu jsou veškerá data uložena a plně dostupná v cílových systémech [24]. Tato definice přesně vystihuje význam termínu databázové migrace, jedná se však o velmi zjednodušený pohled na věc. Později se však v této kapitole dozvíme, že se jedná o proces, který obsahuje několik složitějších kroků a není tak úplně jednoduchý, jak se může na první pohled zdát.

Využití služby pro migraci databázových systémů může mít spoustu příčin. Jedním z hlavních důvodů je šetření nákladů na provoz. Provoz vlastní databáze může být velmi nákladný a to z důvodu vlastní IT infrastruktury, která například zahrnuje výkonný server pro provoz databáze, zakoupení licencí pro databázový systém a dalších nástrojů a zaměstnávání IT pracovníků. Proto se velmi často migruje na databázové systémy, které jsou hostovány v cloudových službách [25] jako je Amazon web services. Další z příčin migrování může být přesun z jedné technologie na druhou. K tomu může dojít, když používáme například starší systém, jiný výrobce nabízí výhodnější cenu licence nebo jiný systém nabízí funkce, které splňují naše nové požadavky [26]. Jako poslední důvod, který můžeme uvést je spojení více databází na jeden server, což nám zlepší přístup k datům a zjednoduší správu databázové infrastruktury [26].

Při migraci databází máme na výběr, kde budeme naše data přemisťovat. Cílem může být například server v lokální síti s nainstalovanou databázovou technologií nebo se může jednat o migraci dat do cloudové služby. Poslední možností je využití kontejnerizace jako je například nástroj Docker. V tomto případě vytvoříme tzv. image, který můžeme libovolně distribuovat a spouštět na jakémkoliv serveru, který má nainstalovaný Docker.

5.3 Postup migrování

Věnujme se nyní samotnému postupu migračního procesu a jednotlivým rozhodnutím, které musíme učinit. Prvním krokem by měla být příprava. Zde bychom měli učinit první rozhodnutí o tom, jak chceme data přemístit, jaký je náš cílový systém, zjistit náklady migrace, vybrat nástroje [25] a zjistit jaká je kardinalita naší migrace. Nejběžnější je 1:1, tedy migrace z jedné zdrojové databáze do jedné cílové databáze, ale existuje také kardinalita 1:N, N:1 a N:M [24]. Je také velmi důležité se seznámit s databázovým schématem zdrojové databáze a jestli je nutné provést jeho konverzi. Ta je nutná během tzv. heterogenní databázové migraci, kdy přemisťujeme data mezi rozdílnými technologiemi, například z MySQL do PostgreSQL nebo z MySQL do MongoDB. Během heterogenní migrace nemusí být konverze schématu vždy nutná, zejména když se jedná o technologie využívající stejný datový model [24]. Jako příklad lze uvést přesun dat z MongoDB do Elasticsearch, oba tyto systémy používají pro ukládání dat JSON dokumenty, nemusíme tedy měnit databázové schéma.

Při homogenní databázové migraci ke konverzi nedochází. Jakmile dokončíme proces přípravy, můžeme pokračovat druhým krokem, což je samotná migrace dat. Pro jejich přesun lze použít různé přístupy jako je třeba bulk insert ze souboru, přesun po dávkách z jedné databáze do druhé, pomocí vytvořených záloh nebo použití externího nástroje. Pokud provádíme migraci po dávkách, je vhodné zamknout databázi pouze pro čtení, aby nedocházelo ke změně již přenesených informací. Po dokončení přenosu dat se dostáváme k poslední částí migračního procesu. Tím je testování cílové databáze. Úkolem je ověřit, že veškeré informace byly úspěšně přeneseny, jsou v korektním stavu a žádné nechybí [27].

5.4 Benefity migrování databází

Nyní se podíváme na jednotlivé výhody migrování databází. Jejich znalost je velmi důležitá, zejména pokud jsme správci databázové infrastruktury v nějaké organizaci. Migrace nám totiž může, jak později uvidíme, pomoci v mnoha ohledech správy databází. Výhody migrace databázových systémů jsou následující:

- **redukce nákladů** - migrací do cloudových služeb jako jsou Amazon web services nebo Microsoft Azure ušetříme náklady na provoz vlastní databázové infrastruktury
- **vhodnější platforma** - přechod na platformu, která je lepší pro naši business aktivitu
- **modernizace software** - migrací na novější verzi používané databázové technologie může mít kladný vliv na výkon a zabezpečení databází
- **zvýšení dostupnosti** - při přesunu dat z jedné databáze na více dalších databází (kardinalita 1:N) dojde ke zvýšení dostupnosti našich dat

Shrnutí

Mezi nejdůležitější poznatky této kapitoly patří zejména definice samotného procesu migrace databázového systému, jeho využití a možných cílových platformách a technologiích. Dále je to samotný postup migrace a jednotlivá rozhodnutí, která musíme učinit. Ty se dají shrnout do tří bodů:

1. **Příprava migrace**
2. **Migrace dat**
3. **Testování cílové databáze**

Zapomenout bychom neměli ani na výše uvedené benefity, které nám mohou pomoci při rozhodování, zda migrovat naši databázi nebo ne.

Kapitola 6

Navyšování výkonu

V poslední části diplomové práce se zaměříme na problematiku navyšování výkonu databázových systémů. V úvodních podkapitolách se podíváme na příčiny a události, které nás mohou vést k řešení problémů s výkonem naší databáze a existujícími nástroji, které můžeme pro jejich vyřešení použít. Dále se budeme zabývat možnostmi navyšování výkonu databázového systému MySQL z hlediska modifikace systémových proměnných, monitoringem metrik a vertikální škálovatelností serveru (fyzického, virtuálního a běžícího v cloudu). Poslední část kapitoly bude věnována výsledkům testování výkonu databáze pro různá nastavení systémových proměnných a velikosti hardwarových zdrojů virtuálního počítače.

6.1 Důvody navyšování výkonu

Během chodu informačního systému nebo jakékoliv aplikace využívající databázový systém pro manipulaci s daty (operace ukládání, načítání, úprava) může nastat okamžik, kdy výkon databáze již není dostačující a stává se z ní tzv. bottleneck celého systému. Ten se může projevit během provozu například při postupně větší návštěvnosti webu a objemu dat. Server tak ztrácí na výkonu s původním nastavením a hardwarovými prostředky při zpracování většího množství požadavků na větším množství dat.

Dále lze bottleneck odhalit před samotným nasazením aplikace nebo informačního systému pomocí správného testovacího procesu. Pro správné otestování výkonu je potřeba dobrý odhad běžného a vysokého zatížení serveru. Tím je myšleno například počet připojených uživatelů a počet dotazů zasílaných na databázi. Jako příklad různého vytížení lze uvést mnohonásobně větší počet zákazníků e-shopů během adventního období oproti všedním dnům. Více připojených uživatelů tedy vyprodukuje více dotazů, které musí databáze zpracovat a bez správného navýšení výkonu databázové infrastruktury by mohlo dojít ke zpomalení a v nejhorším případě i výpadku celého systému.

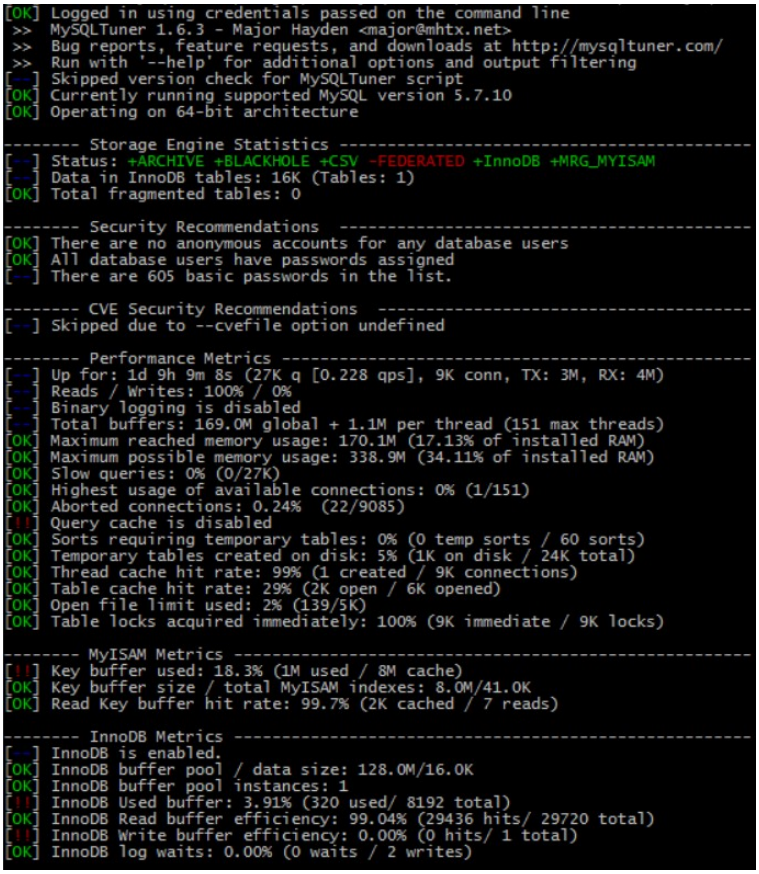
Můžeme tedy vidět, že hlavním důvodem pro zvýšení výkonu databáze je zajištění bezproblémového chodu i při neobvykle vyšším zatížení, zvyšujícím se objemu dat a vyčerpání hardwarových

zdrojů systému. V následující kapitole se podíváme na existující software, který nám může pomoci při řešení problémů s výkonem databázového serveru.

6.2 Existující software

Pro řešení problému s výkonem databázového systému MySQL existuje spousta komerčního a nekomerčního software, který můžeme použít. Mnoho z nich je ve formě obyčejného skriptu, který lze pohodlně spustit i z unixového shellu a některé jsou propracované nástroje s rozsáhlým grafickým uživatelským rozhraním. V následujících řádcích si představíme aplikace, které spadají do obou těchto skupin.

Začněme skriptem, který je napsaný v programovacím jazyce Perl a tím je MySQLTuner [28]. Ten se po spuštění připojí k databázi pomocí zadaných přihlašovacích údajů, zkontroluje připojení a zda má dostatečná uživatelská práva. Poté provede analýzu metrik dané MySQL instance a následně zobrazí výpis jejího stavu a doporučení pro zvýšení výkonu. Kromě výkonnostní analýzy také provádí jednoduchou analýzu zabezpečení z hlediska uživatelských účtů a hesel.



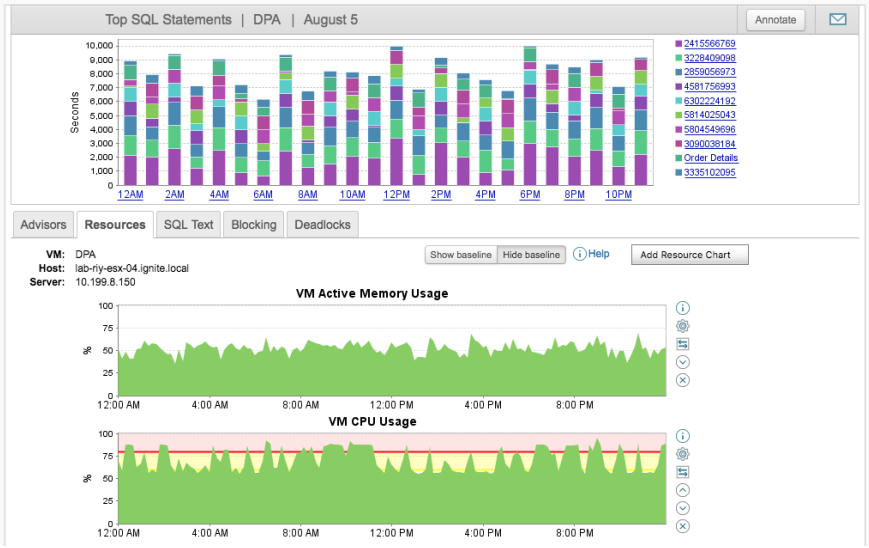
Obrázek 6.1: MySQL Tuner [28]

Dalším ze zástupců skriptů je Tuning-Primer [29], který je napsaný ve skriptovacím jazyce Bash. Pomocí informací o systémových proměnných a systémových metrikách poskytuje uživateli sadu doporučení, které nám pomohou se správnou konfigurací serveru. Zmínit můžeme také program Mysqlreport, který rovněž funguje na podobném principu.

Prvním nástrojem s grafickým uživatelským rozhraním, který si představíme, je Performance tools, který je součástí známé aplikace MySQL Workbench pro správu MySQL instance. V něm jsou nám k dispozici statistiky a grafy ohledně stavu sítě (počet připojení, velikost vstupního a výstupního provozu), aktivity serveru a stavu InnoDB engine (využití bufferu, velikost zápisu a čtení z disku). Dále nám poskytuje rozsáhlé informace o vykonaných dotazech a vizualizaci plánu vykonání dotazu.

Jedním z nejpoužívanějších programů pro zlepšení výkonu databázového serveru je SQL Diagnostic Manager for MySQL od společnosti Webyog [30]. Ten nám dává k dispozici přes 600 různých monitorů, pomocí kterých můžeme lehce sledovat vývoj naší instance, správně ladit problematické dotazy a pomáhá nám se rozhodnout, které kroky je třeba učinit pro zlepšení výkonu. Všechny tyto monitory a s nimi spojené statistiky jsou dostupné v reálném čase.

Zmínit můžeme také nástroj Database performance analyzer [31] od společnosti Solarwinds, který nám také poskytuje analýzu výkonu databázového serveru zejména z hlediska využití paměti, procesoru a vykonávání dotazů. Jako poslední software lze uvést dbForge studio, jehož hlavním cílem je ladění dotazů.



Obrázek 6.2: Solarwinds Database Performance analyzer [31]

Nyní, když už známe existující řešení v oblasti navyšování výkonu databázového serveru, je na čase, abychom jsme se podívali na jednotlivé možnosti, jak zlepšit výkon MySQL serveru. To bude cílem následující podkapitoly.

6.3 Navyšování výkonu MySQL

Pro zlepšení výkonu databázového serveru MySQL můžeme provést množství konfigurací jak databázové instance, tak samotného hostitelského počítače. Začněme tedy tím, jak můžeme pozitivně ovlivnit výkon databáze modifikací serveru, na kterém běží. V oblasti hardwaru nás budou zajímat tři části a to je procesor, operační paměť RAM a typ úložiště. Procesor má na starost zejména vykonávání dotazů, vybírání nejlepšího plánu vykonání a celkově zpracování veškerých požadavků a uživatelských připojení. Je tedy zřejmé, že se něj kladou vysoké nároky (zejména u složitých dotazů provádějící matematické výpočty) a s narůstající zátěží jsou čím dál vyšší. Pořízením výkonnějšího procesoru, nebo zvýšením počtu výpočetních jader v případě virtuálního stroje, můžeme výrazně zlepšit výkon databáze.

Podobně jako CPU, má i operační paměť velký vliv na rychlost zpracování dotazů. Je to z toho důvodu, že databázový systém v této paměti alokuje část, která se nazývá buffer a do ní si načítá data z tabulek a indexů, se kterými momentálně pracuje. Důvodem je rychlost operací čtení a zápisu v hlavní paměti, které jsou až 1000krát rychlejší než na disku [32]. V ideálním případě je tedy vhodné mít na hostitelském počítači dostatečně velkou operační paměť, do které můžeme načíst většinu dat a omezit tak nutnost čtení z disku, při zpracovávání dotazů, na minimum.

Poslední hardwarová součást serveru, která má vliv na výkon databáze, je typ použitého úložiště. Z hlediska rychlosti diskových operací je SSD disk mnohonásobně rychlejší (až 10krát), než pevný disk, ale je také přibližně 2krát dražší a kapacitně menší. Když ale porovnáme cenu s poskytovaným výkonem, tak je SSD rentabilní pro většinu případů. HDD je vhodné pouze pro situace, kdy máme data o velikosti větší než 10 TB a nepřistupuje se k nim velmi často [33].

Na úrovni operačního systému serveru, který hostuje databázi, je vhodné nakonfigurovat proces, který se nazývá swapování. Jeho účelem je přemístění neběžících procesů z hlavní paměti počítače na disk, aby došlo k uvolnění paměti pro nové procesy. Když se neaktivní proces opět aktivuje, tak je načten z disku zpátky do operační paměti. Tyto I/O operace mohou zpomalit celý systém, proto je důležité, aby k nim docházelo ojediněle. To lze zařídit pomocí nastavení sysctl -w vm.swappiness=1 v operačním systému Linux.

Nestačí mít pouze správný hardware jako je dostatek paměti RAM nebo výkonný procesor. Důležitá je i správná konfigurace samotné databázové instance. Ta se provádí pomocí úprav systémových proměnných jako jsou například velikosti různých bufferů nebo maximální počet připojení. Jejich aktuální stav si můžeme zobrazit pomocí dotazu SHOW GLOBAL VARIABLES a pro nastavení jejich hodnoty SET GLOBAL název = hodnota. V systému MySQL jsou nám k dispozici metriky, které můžeme sledovat a na základě jejich velikostí správně upravit hodnoty proměnných. Pro zobrazení metrik slouží příkaz SHOW GLOBAL STATUS. Existují stovky různých proměnných a metrik [15], které můžeme v systému MySQL monitorovat a konfigurovat. My se v následujícím seznamu podíváme na některé z nich.

- **thread_cache_size** - udává, kolik vláken bude uloženo do cache pro znovupoužití. Je vhodné zvětšit tuto cache v případě, že se vytváří spoustu nových připojení a vláken. To lze monitorovat pomocí metriky Threads_created
- **sort_buffer_size** - buffer, který se využívá během operací ORDER BY a GROUP BY. Pokud je hodnota metriky Sort_merge_passes vysoká, je vhodné zvýšit hodnotu tohoto bufferu.
- **join_buffer_size** - tato paměť se používá v případě, kdy operace join nepoužívá index a provádí tak operaci full table scan. Používá se zejména během algoritmů nested loop join a batched key access. Zvýšení velikosti tohoto bufferu je vhodné pouze v rámci dané session, která provádí spoustu rozsáhlých join operací a přidání nových indexů není možné.
- **read_buffer_size** - jedná se o paměť, která se používá u MyISAM engine a operace sekvenčního čtení. Pro každou tabulku se alokuje buffer o dané velikosti. Pokud je hodnota metriky Select_scan velká, tak to znamená, že se provádí spousta operací sekvenčního čtení a je třeba navýšit hodnotu read_buffer_size.
- **tmp_table_size** - tato proměnná nám říká, jaká je maximální velikost dočasných tabulek v hlavní paměti. Pokud dočasná tabulka překročí tento limit, tak ji databázový systém přemístí z hlavní paměti na disk. Sledováním Created_tmp_disk_tables a Created_tmp_tables můžeme zjistit, zda je potřeba navýšit jejich maximální velikost.
- **innodb_buffer_pool_size** - jedná se o velikost hlavního bufferu databázového engine InnoDB, do kterého si načítá data tabulek a indexů. Větší hodnota této proměnné tedy znamená menší počet I/O operací na disku. Na serverech, které hostují pouze MySQL instanci, můžeme hodnotu InnoDB bufferu nastavit až na 80% velikosti operační paměti počítače. Pomocí vzorečku innodb_buffer_pool_reads/innodb_buffer_pool_read_requests*100 lze vypočítat účinnost bufferu a zjistit tak, jestli je jeho velikost dostatečná.
- **table_open_cache** - představuje limit počtu tabulek, které mohou být otevřeny pro všechna vlákna. Pokud dochází k otevírání spousty tabulek (metrika Opened_tables), je pro zvýšení výkonu potřeba navýšit hodnotu této proměnné.
- **key_buffer_size** - tento buffer se používá pouze s databázovým engine MyISAM a slouží pro uložení bloků indexu. Jestliže často dochází ke čtení indexu z disku, což poznáme podle velké velikosti metriky Key_reads, je nutné přiřadit bufferu více paměti. Doporučená velikost, podle oficiální MySQL dokumentace, je 25% operační paměti počítače.
- **max_connections** - definuje maximální počet současných připojení k serveru. Když podle hodnoty Connection_errors_max_connections zjistíme, že často dochází k odmítání nových připojení z důvodu dosažení maximální kapacity, je vhodné zvýšit počet možných připojení a dovolit tak uživatelům pracovat s databází. Zde ale musíme být opatrní, protože spousta

systémových proměnných jsou vázána na jedno sezení. Lehce tak můžeme neohleduplným zvýšením počtu připojení vyčerpat hardwarového zdroje serveru a způsobit tak jeho zpomalení, případně i pád celého systému.

Můžeme vidět, že modifikací několika systémových proměnných lze jednoduše zvýšit výkon databázového serveru, zejména pokud se jedná o proměnné, jejichž úpravou zvýšíme počet operací v hlavní paměti počítače a snížíme počet diskových operací.

Poslední možností navýšení výkonu databáze, kterou v této práci zmíníme, je ladění SQL příkazů. Základním a nejčastěji používaným krokem, je zobrazení plánu vykonání dotazu. Jeho analýzou lze snadněji pochopit, jaké operace databázový systém během vykonávání příkazu provádí a jaká byla jejich cena (ta je stanovena matematickým výpočtem podle použitých zdrojů CPU, paměť a I/O). Na základě této analýzy můžeme následně například vytvořit index pro atribut tabulky, kvůli kterému docházelo k operaci sekvenčního průchodu během spojování tabulek. Po jeho vytvoření se při vykonání dotazu použije operace přímého přístupu, která je efektivnější, jelikož zbytečně neprochází data, která nejsou pro daný dotaz důležitá. Dále můžeme využít techniky přepisu, kdy pro daný dotaz vytvoříme ekvivalentní zápis, který nám vrátí stejná data, ale query optimizer (který se stará o generování plánů vykonání a výběr nejlepšího plánu) vytvoří a vybere jiný plán vykonání dotazu, který může být efektivnější. Při výběru plánu se query optimizer řídí statistikami, které se vytvářejí pro indexy a jednotlivé atributy v tabulkách. V nich je například uveden počet řádků v tabulce, počet odlišných hodnot ve sloupcích nebo jaká je distribuce dat. Tyto statistiky nám v databázovém systému slouží k vytvoření odhadu kardinality jednotlivých operátorů v plánu vykonání a pomocí tohoto odhadu vypočítat cenu operátoru. Když jsou operátory takto naceněny, je query optimizer schopen vybrat nejlevnější a tím pádem nejefektivnější plán vykonání dotazu. Může se ale stát, že pro daný atribut, se kterým v dotazu pracujeme, neexistuje vhodná statistika (například distribuce dat v daném sloupci), nebo obsahuje špatné údaje [34] a query optimizer tak musí při vytváření odhadu a ceny operátoru použít vlastní předpoklad. To může mít za následek vybrání méně efektivního plánu. Během ladění dotazů můžeme vytvořit nové statistiky a tím ovlivnit, jaký plán vykonání dotazu query optimizer vybere.

Můžeme tedy vidět, že existuje spousta možností, jak zvýšit výkon databázového serveru. Ale ne vždy je nutné sáhnout po všech výše zmíněných variantách. Je jen na naší odborné znalosti abychom nalezli výkonnostní problém a zvolili správné řešení pro dosažení optimálního výkonu.

Všechny zmíněné optimalizačních techniky fungují samozřejmě také pro databázové instance, které běží v prostředí libovolného poskytovatele cloudových služeb, jako je například AWS. Amazon web services nám v rámci služby relačních databází poskytuje své vlastní nástroje pro monitorování a navyšování výkonu databáze. Jedná se o nástroje Performance Insights a Enhanced monitoring. Z hlediska vertikálního škálování (zvýšení CPU, paměti. . .) je nám k dispozici mnoho různých typů (ty představují různé kombinace CPU, paměti a kapacity sítě) a úrovní instancí (počet hardwarových zdrojů). Můžeme si tak například vybrat typ T3, který je určen pro obecné použití s úrovní instance

db.t3.micro, která poskytuje 1 GB paměti a 2 CPU. Až budeme potřebovat zvýšit velikost hlavní paměti, nebo počet procesorů, stačí nám pouze přejít na vyšší úroveň instance.

6.4 Testování výkonu

Ohledně výkonu databázového systému MySQL jsme provedli několik testovacích příkladů, ve kterých testujeme vliv konfigurace jednotlivých bufferů systému a počtu výpočetních jader procesoru na rychlost vykonávání SQL příkazů. Předpokladem těchto testů je, že zvýšení velikosti testovaných bufferů bude mít pozitivní vliv na výkon databáze. Pojďme se tedy podívat na jednotlivé výsledky.

První měření zkoumalo rychlost vykonání dotazu obsahující operaci třídění. Manipulovali jsme tedy se systémovou proměnnou sort_buffer_size, která udává velikost bufferu, který využívá jak databázový engine MyISAM, tak i engine InnoDB během třídění dat.

Tabulka 6.1: Testování sort_buffer_size

Pokus #	262,14 kB	524,29 kB
1	99,19s	45,38s
2	67,41s	41,61s
3	56,40s	41,46s
4	69,66s	41,99s
5	65,51s	40,10s
6	84,64s	45,72s
7	69,19s	41,84s
8	64,14s	40,77s
průměr	72,02s	42,36s

Z tabulky 6.1 uvedené výše můžeme vyčíst, že zvýšení hodnoty tohoto bufferu ze základních 262 kB na dvojnásobnou hodnotu, zrychlilo vykonání dotazu v průměru o 30 sekund. V tomto případě jsme tedy dosáhli požadovaného cíle.

Další testovanou systémovou proměnnou je innodb_buffer_pool_size, kterou používá pouze engine InnoDB a to pro ukládání právě zpracovávaných dat tabulek a indexů. První sada měření proběhla opět na defaultní hodnotě testované proměnné. V tomto případě se jedná o 128 MB. Hodnota bufferu byla pro druhou sadu testů výrazně zvýšena a to na 4 GB.

Tabulka 6.2: Testování innodb_buffer_pool_size

Pokus #	128 MB	4 GB
1	41,83s	16,28s
2	50,00s	16,20s
3	46,62s	16,50s
4	46,77s	16,07s
5	43,46s	15,95s
6	49,87s	15,30s
7	43,26s	16,70s
8	43,71s	15,10s
průměr	45,69s	16,01s

Opět můžeme vidět, že zvětšení velikosti bufferu mělo za následek obrovské zrychlení při vykonávání dotazů, jelikož jsme umožnili databázovému systému si načíst větší porci dat do operační paměti a omezili tak počet pomalejších diskových operací.

Následující dvě proměnné využívá pouze engine MyISAM a jedná se o key_buffer_size (paměť pro bloky indexu) a read_buffer_size (paměť využívaná během sekvenčního čtení). Velikost proměnných v první sadě měření byla nastavena na úplné minimum, což je velikost jedné stránky (8 kB) a pro druhou sadu jsme ji zvýšili na 262 kB.

Tabulka 6.3: Testování bufferů MyISAM engine

(a) key_buffer_size

Pokus #	8,192 kB	262,144 kB
1	117,88s	103,85s
2	107,66s	103,05s
3	112,92s	102,77s
4	106,36s	102,79s
5	106,87s	102,80s
6	109,94s	103,04s
7	115,00s	111,34s
8	114,18s	103,76s
průměr	111,35s	104,18s

(b) read_buffer_size

Pokus #	8,192 kB	262,144 kB
1	102,46s	93,03s
2	96,25s	93,01s
3	111,14s	93,76s
4	98,06s	92,81s
5	107,08s	103,42s
6	108,87s	95,68s
7	96,31s	92,49s
8	102,24s	95,20s
průměr	102,80s	94,93s

Z výsledků testů uvedených v tabulce 3 a 4 jde vidět, že provedené změny měly opět očekávaný výsledek, tedy došlo ke zrychlení výkonu databáze. Všechny čtyři provedené testy měly pozitivní

vliv na rychlost databázového systému a potvrdily tak náš předpoklad. Můžeme tedy usoudit, že systémové proměnné mají velký dopad na výkon celého systému a jejich správné nastavení je základem pro výkonnou databázi.

Kromě velikosti bufferů jsme měřili také vliv počtu výpočetních jader procesoru na rychlost vykonání dotazů. K testování jsme využili virtuální stroj s operačním systémem Ubuntu, kterému jsme postupně nastavili 2, 3 a 4 výpočetní jádra.

Tabulka 6.4: Testování počtu procesorů

Pokus #	2 CPU	3 CPU	4 CPU
1	93,03s	79,27s	73,41s
2	93,01s	76,89s	76,42s
3	93,76s	73,81s	63,74s
4	92,81s	75,21s	74,81s
5	103,42s	77,57s	62,70s
6	95,68s	74,10s	65,32s
7	92,49s	77,60s	73,46s
8	95,20s	80,26s	73,72s
průměr	94,93s	76,84s	70,45s

I pro naše menší zatížení můžeme pozorovat nárůst výkonu databázového systému s narůstajícím počtem procesorů, které měl počítač k dispozici.

Jednoduchými testy jsme si potvrdili platnost některých možností navyšování výkonu databázového systému MySQL uvedených v předchozí části této kapitoly. Následující kapitoly diplomové práce budou již zaměřeny na implementační detaily praktické části.

Shrnutí

V rámci této kapitoly jsme se dozvěděli o důležitosti navyšování výkonu databázové části informačního systému a o jednotlivých možnostech, které můžeme pro zrychlení databáze použít. Ve zkratce je můžeme shrnout následovně:

1. modifikace hostitelského počitače
   (a) hardwarová - zvýšení hardwarových prostředků, pořízení výkonnějších součástek
   (b) softwarová - nastavení na úrovni operačního systému (například swapování)
2. konfigurace databázových proměnných - velikosti bufferů, cache, počet připojení atd.
3. ladění dotazů

(a) vytváření indexů

(b) přepis dotazů

(c) vytváření statistik

Na konci této kapitoly jsme si ukázali výsledky testů, které demonstrovaly platnost výše uvedených možností navyšování výkonu.

Kapitola 7

Implementace

Na začátku této kapitoly se podíváme na použité technologie pro implementaci jednotlivých nástrojů a testů. Jmenovitě se bude jednat o použitý operační systém, programovací jazyk, balíčky a knihovny, databázové systémy a nástroje a služby pro provoz virtualizace, kontejnerizace a cloudových služeb. Druhá část kapitoly bude věnována popisu jednotlivých implementací provedených v rámci této diplomové práce.

7.1 Použité technologie

Nyní se tedy podíváme na technologie, které jsme v rámci implementace nástrojů a testů pro zálohování, migraci databází a navyšování výkonu použili.

Použitý operační systém

Pro běh jednotlivých databází byl použit operační systém Linux a jeho distribuce Ubuntu ve verzi 20.04. Hlavní důvody zvolení tohoto operačního systému a konkrétní distribuce jsou následující:

- **Open-source a bezplatný** - nemusíme kupovat žádnou drahou licenci, což je obrovskou výhodou nejen pro naše studijní účely, ale i pro produkční prostředí.
- **Systém balíčků** - Ubuntu obsahuje správce balíčků *apt*, který usnadňuje správu softwaru a díky němu je instalace nových programů a služeb rychlá a jednoduchá.
- **Zabezpečení systému** - Linuxové distribuce jsou známé pro svou vyšší odolnost vůči virovým útokům a menším počtem "děr"v zabezpečení systému.
- **Stabilita systému** - stabilní verze Ubuntu obsahují minimum chyb a jsou velmi robustní.

Programovací jazyk

Pro implementaci jednotlivých nástrojů byl zvolen vysokoúrovňový skriptovací programovací jazyk Python. Tento jazyk byl zvolen zejména kvůli počtu dostupných knihoven pro práci s databázemi, službou docker, manipulaci s daty ve formátu JSON, jednoduchou komunikaci s unixovým shellem a na základě osobní preference. Dále je tento jazyk také multiplatformní, vytvořené nástroje jdou tedy spouštět na jakékoliv platformě. Během implementace výkonnostních testů zálohování a obnovy databází byl použit i programovací jazyk Bash, který zjednodušil spouštění databázových nástrojů pro zálohování a obnovu.

Použité balíčky a knihovny

Abychom v jednotlivých skriptech mohli pracovat s databázemi, musíme mít nainstalované potřebné knihovny. Knihovnám, které slouží k práci s databázemi (přípojení, zasílání dotazů, získávání dat apod.) se říká ovladač nebo konektor. K manipulaci s databází MySQL jsme použili knihovnu *mysql.connector*.

```
import mysql.connector

connection = mysql.connector.connect(
  host="localhost",
  user="uzivatel",
  password="tajneheslo",
  database="databaze"
)

cursor = connection.cursor()

cursor.execute("SELECT * FROM uzivatele")

vysledek = cursor.fetchall()
print(vysledek)

# Vystup
# [(1, 'Jan', 'Novak'), (2, 'Karel', 'Dvorak')]
```

Listing 7.1: Ukázka práce s knihovnou mysql.connector

V ukázce kódu 7.1 můžeme vidět základní práci s konektorem pro MySQL, konkrétně vytvoření nového připojení k databázi, vytvoření kurzoru, vykonání dotazu a získání výsledku. Dalším použitým

ovladačem je *pymongo* pro databázi MongoDB. Práce s nim je jednoduchá, po vytvoření nového spojení se vybere databáze a následně kolekce, viz následující ukázka 7.2.

```
import pymongo

connection = pymongo.MongoClient("mongodb://localhost:27017/")
databaze = connection["database"]
kolekce = databaze["uzivatele"]

vysledek = kolekce.find_one()

print(vysledek)

# Vystup
# {'id': 1, 'jmeno': 'Jan', 'prijmeni': 'Novak'}
```

Listing 7.2: Ukázka práce s knihovnou pymongo

Posledním použitým databázovým ovladačem je *elasticsearch* pro systém Elasticsearch. Jeho použití je velmi podobné jako u předchozích dvou ovladačů.

```
from elasticsearch import Elasticsearch

connection = Elasticsearch([{'host': 'localhost', 'port': 9200}])

vysledek = connection.search(index='uzivatele', body={
                "query": {"match_all": {}}, "results": limit})
print(vysledek["hits"]["hits"][0]["_source"])

# Vystup
# {'id': 1, 'jmeno': 'Jan', 'prijmeni': 'Novak'}
```

Listing 7.3: Ukázka práce s knihovnou *elasticsearch*

Nástroj pro migrování databází provádí migraci do Docker kontejneru. K uskutečnění takové migrace je zapotřebí knihovna *docker*, která nám poskytuje aplikační rozhraní k nainstalovanému nástroji Docker na našem počítači. Pomocí tohoto rozhraní pak můžeme například vytvářet nové kontejnery nebo manipulovat s již existujícími kontejnery.

```
import docker

klient = docker.from_env() # získání instance Docker z prostředí operačního systé
    mu
```

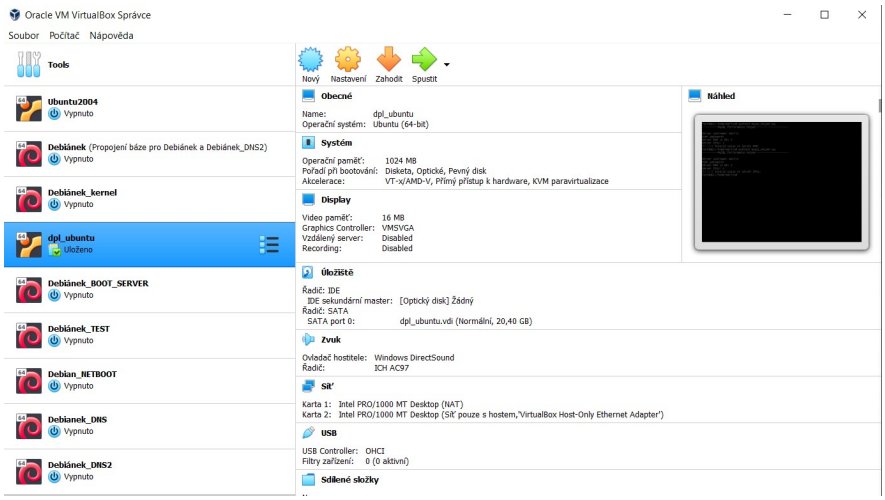
```
klient.images.build(path="cesta/k/dockerfile", tag=mujImage) # Vytvoření nového
    Docker Image
klient.containers.run(mujImage, name=mujKontejner, detach=True) # Vytvoření nového
    kontejneru
```

Listing 7.4: Ukázka práce s knihovnou *docker*

Zmínit můžeme také knihovny *os* pro využití funkcí závislých na systému jako například spuštění příkazů unixového shellu, *colorama* pro přidání barev do textových výpisů a *getpass*, která poskytuje funkci pro bezpečné zadávání hesel z příkazové řádky.

Virtualizace

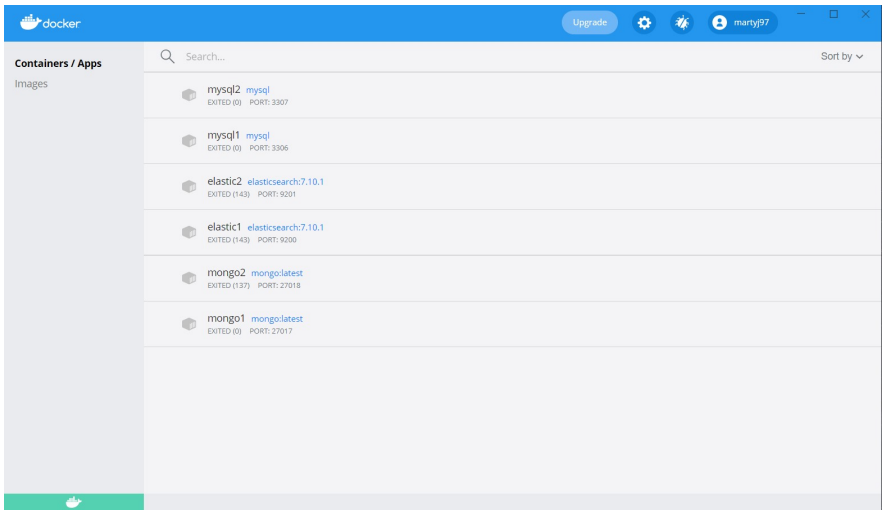
Jak již bylo v kapitole 6 o navyšování výkonu zmíněno, tak jednou z možností zlepšení výkonu databáze je pořízení lepšího hardwaru. K demonstraci této možnosti byla použita virtualizace. Mohli jsme tak libovolně měnit velikost operační paměti a počtu procesorů a testovat tak rychlost databáze pro jednotlivé hardwarové konfigurace. Pro virtualizační proces byl použit volně dostupný open-source nástroj VirtualBox od společnosti Oracle.



Obrázek 7.1: Prostředí nástroje VirtualBox

Kontejnerizace

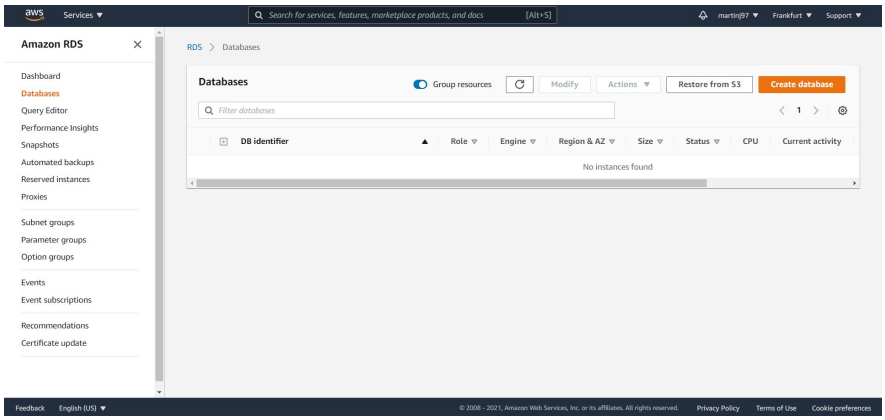
Kontejnerizace byla využita v kapitole ohledně migrace databází, kde v implementovaném nástroji můžeme zvolit kontejner jako jednu z možných destinací pro migrovanou databázi. Detailní popis kontejnerů je uveden v kapitole 2.1. O vytváření a správu kontejnerů se stará nejrozšířenější program v oblasti kontejnerizace Docker ve verzi Desktop, která je určena pro operační systém Windows 10.



Obrázek 7.2: Prostředí nástroje Docker Desktop

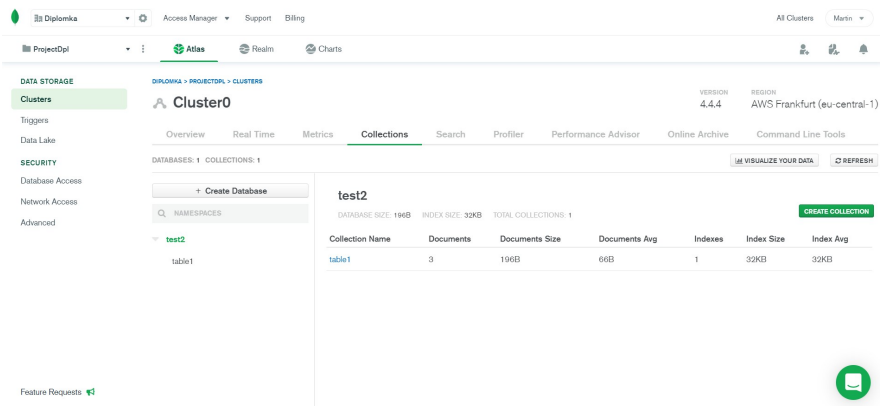
Cloudové služby

Nejčastějším cílem databázové migrace jsou cloudové služby, které poskytují databázové systémy. Implementovaný nástroj proto podporuje migraci do těchto služeb. Jmenovitě se jedná o AWS, MongoDB Atlas a Elastic Cloud. Amazon Web Services jsme použili pro migraci databázového systému MySQL. Zde máme v bezplatné verzi 750 výpočetních hodin, 20 GB místa na SSD disku, 1 GB RAM a 1 vCPU.



Obrázek 7.3: Prostředí Amazon Web Services

MongoDB Atlas poskytuje pouze 512 MB úložného prostoru v bezplatné verzi pro hostování databáze MongoDB, ale doba užívání není nijak časově omezena. Elastic Cloud je z těchto tří cloudových služeb nejvíce omezen v bezplatné verzi. Ta je dostupná pouze na 14 dnů. V rámci diplomové práce byly vybrány právě tyto bezplatné verze, jelikož byly pro testovací účely dostačující.



Obrázek 7.4: Prostředí MongoDB Atlas

Databázové systémy

V diplomové práci jsme pracovali se třemi databázovými systémy. Jedná se o relační databázi MySQL, NoSQL dokumentovou databázi MongoDB a Elasticsearch. Tyto systémy byly vybrány záměrně s cílem ukázat, vyzkoušet a otestovat zálohování a obnovu různých typů databázových systémů a poukázat na úskalí během migrace mezi heterogenními databázemi.

7.2 Implementace výkonnostních testů zálohování

MySQL

Nejprve se podíváme na implementaci zálohování databází a testů rychlosti vytváření záloh. Pro databázový systém MySQL jsme vytvářeli nejdříve úplnou zálohu. K vytvoření úplné zálohy nám slouží následující skript *mysql_backup.sh*, který má tři parametry. Ty jsou uživatelské jméno, heslo a databáze, kterou chceme zálohovat.

```
#!/bin/bash

SECONDS=0

mysqldump --single-transaction -u $1 -p$2 $3 > `date +"%Y-%m-%d"`.sql

dur=$SECONDS

echo "$(($dur)) seconds."
```

Listing 7.5: Vytvoření úplné zálohy MySQL

Skript vytvoří soubor se zálohou ve formátu .sql a s názvem aktuálního data (rok-měsíc-den) a vypíše se doba trvání vytvoření úplné zálohy v sekundách . Nastavení *–single-transaction* zajistí, že záloha databáze bude korektní a nedojde k uzamčení tabulek a tím pádem se nepřeruší operace čtení a zápisu ostatních připojení. Tuto volbu lze použít pouze pro tabulky, které používají engine InnoDB. Pro engine MyISAM je potřeba použít nastavení *–lock-tables=false*, které zapříčiní, že nedojde k uzamčení tabulek zámkem pouze pro čtení (READ LOCK).

Inkrementální zálohování v MySQL je implementováno pomocí binárních logů. Do nich se ukládají veškeré operace, které manipulují s daty. Zálohou těchto logů tedy provedeme inkrementální zálohování databáze.

```
#!/bin/bash

SECONDS=0

sudo mysql -E --execute='FLUSH BINARY LOGS;' mysql

cp /data/mysqlbin_back/mysql-bin.* /data/bin_back

last=ls /data/bin_back | tail -n -1

rm /data/bin_back/$last

sudo mysql -E --execute "purge binary logs to $last;" mysql

dur=$SECONDS

echo "$(($dur)) seconds."
```

Listing 7.6: Vytvoření inkrementální zálohy MySQL

Výše uvedený skript 7.6 zkopíruje binární logy do cílové složky, čímž provede jejich zálohu. Před samotným kopírováním se spustí MySQL dotaz FLUSH BINARY LOGS, který způsobí to, že se začne psát do nového souboru. Po zkopírování se vybere poslední soubor (nově vytvořený) a ten se ze zálohy odstraní. Následně všechny zálohované binární logy odstraníme z původní složky logů pomocí dotazu PURGE BINARY LOGS. V rámci inkrementální zálohy se v pravidelných intervalech (např. 1 týden) provádí také úplná záloha podobným způsobem, jaký je uveden ve výpisu 7.5. Akorát musíme přidat nastavení *–flush-logs* a *–delete-master-logs*, aby došlo k odstranění binárních logů. Pokud bychom tak neučinili, při další inkrementální záloze by došlo k zálohování dat, která jsou již obsažena v úplné záloze. Vytvoření celkové zálohy pak vypadá následovně:

```
mysqldump --flush-logs --delete-master-logs --single-transaction -u $user -
    p$password $database > `date +"%Y-%m-%d"`.sql
```

Listing 7.7: Vytvoření úplné zálohy u inkrementálního zálohování

Obnova databáze pomocí úplné zálohy je velmi jednoduchá. Provádí se následujícím způsobem:

```
#!/bin/bash

SECONDS=0

mysql -u martin -p dpl_restore < /data/mysqldump/dpl.sql

dur=$SECONDS

echo "$(($dur)) seconds."
```

Listing 7.8: Obnovení MySQL databáze pomocí úplné zálohy

Pro obnovení databáze z inkrementální zálohy musíme provést dva kroky. Prvním krokem je obnova úplné zálohy stejným způsobem, jaký je uveden ve výpisu 7.8. Krok druhý je obnova pomocí binárních logů použitím nástroje *mysqlbinlog*. To se dělá následovně:

```
#!/bin/bash

SECONDS=0

cesta=$1
user=$2
pass=$3
db=$4

files=`ls -1 $cesta`

for file in $files
do
  mysqlbinlog $file | mysql -u $user -p$pass $db
done

dur=$SECONDS
```

```
echo "$(($dur)) seconds."
```

Listing 7.9: Obnovení MySQL databáze pomocí inkrementální zálohy

Pro otestování rychlosti vytvoření záloh za běžícího provozu, byl implementován skript, který simuluje několik připojených uživatelů komunikujících s databází. Ten funguje na principu vytvoření několika vláken a zasílání náhodného počtu dotazů na databázi.

```
number_of_queries = random.randint(10, 10000)

for j in range(number_of_queries):
    id = random.randint(1, 3000000)
    limit = random.randint(10, 1000)
    cursor_master.execute(
        "select cache from produkty where id >= %s order by id limit %s", (id,
            limit))
    cursor_master.fetchall()

cursor_master.close()
db_master.close()
```

Listing 7.10: Zátěž databázového systému MySQL

MongoDB

Databázový systém MongoDB má pro vytváření úplných záloh nástroj *mongodump*, který je velmi podobný nástroji *mysqldump*. Jeho výstupem je soubor ve formátu .bson (binary json). Následujícím skriptem vytvoříme úplnou zálohu databáze do nové složky s aktuálním datem.

```
#!/bin/bash

SECONDS=0

mongodump --db mydatabase --out /data/mongobackups/`date +"%Y-%m-%d"`

dur=$SECONDS

echo "$(($dur)) seconds."
```

Listing 7.11: Vytvoření úplné zálohy MongoDB

Obnovení databáze z úplné zálohy se provádí pomocí funkce *mongorestore*. Té předáme jako argument název databáze k obnovení a cestu ke složce obsahující celkovou zálohu databáze.

```
#!/bin/bash

SECONDS=0

mongorestore --db mydatabase --drop /data/mongobackups/2021-03-25/mydatabase

dur=$SECONDS

echo "$(($dur)) seconds."
```

Listing 7.12: Obnovení databáze MongoDB pomocí úplné zálohy

Inkrementální zálohu je v systému MongoDB možno udělat pomocí zálohování kolekce oplog. V ní jsou obsaženy všechny změny provedené v databázi. Jednotlivé záznamy obsahují časové razítko (timestamp), podle kterého můžeme vytvářet jednotlivé zálohy. Následující skripty představují postup pro vytvoření inkrementální zálohy.

```
#!/bin/bash

ts=‘mongo --quiet --eval 'db.oplog.rs.find().sort({$natural:-1}).limit(1).next().
    ts' local‘

mongodump --db mydatabase --out /backups/‘date +"%Y-%m-%d"‘

echo $ts > timestamp.txt
```

Listing 7.13: Úplná záloha v rámci inkrementální zálohy MongoDB

Nejprve si vytvoříme úplnou zálohu databáze pomocí nástroje *mongodump* a do souboru *timestamp.txt* si uložíme poslední časové razítko z kolekce oplog. Od této značky se vytvoří nová inkrementální záloha pomocí následujícího skriptu.

```
#!/bin/bash

ts=‘cat timestamp.txt‘ # ts = Timestamp(1616595521, 1)
t=${ts:10:10} # 1616595521
i=${ts:22:1} # 1

mongodump --db local --collection oplog.rs --out /data/mongodump/oplogdump --query
     "{\"ts\" : { \"\$gte\" : { \"\$timestamp\" : { \"t\" : $t, \"i\" : $i } } }}"
```

```
newts=‘mongo --quiet --eval ’db.oplog.rs.find().sort({$natural:-1}).limit(1).next
    ().ts’ local‘
echo $newts > timestamp.txt
```

Listing 7.14: Vytvoření inkrementální zálohy v MongoDB

Opět použijeme nástroj *mongodump*, kterému předáme argument *–collection* a specifikujeme tak název kolekce. Jako argument je také předán dotaz, kterým říkáme, že se mají zálohovat pouze záznamy od daného časového razítka. Obnova databáze se provádí pomocí funkce *mongorestore* s nastavením *–oplogReplay* a vypadá následovně:

```
#!/bin/bash

mongorestore --oplogReplay /data/mongodump/oplogdump/local
```

Listing 7.15: Obnova z inkrementální zálohy v MongoDB

Podobně jako v případě testování MySQL byl i pro databázový systém MongoDB implementován jednoduchý skript pro simulování provozu na databázi. Funguje na stejném principu jako výše zmíněný skript 7.10 pro zátěž na MySQL databázi.

Elasticsearch

Jak již bylo zmíněno v kapitole 4.5, v systému Elasticsearch se pro vytváření úplných i inkrementálních záloh používají snapshoty. Nejprve musíme vytvořit repozitář, kde se budou jednotlivé snapshoty ukládat. Vytvoření repozitáře a snapshotu vypadá následovně:

```
curl -XPUT "localhost:9200/_snapshot/muj_repozitar" -d ’
{
  "type": "fs",
  "settings":
  {
    "location": "/data/elastic_repo"
  }
}’
```

Listing 7.16: Vytvoření repozitáře v Elasticsearch

```
#!/bin/bash

SECONDS=0
```

```
curl -XPUT "localhost:9200/_snapshot/muj_repozitar/muj_snapshot01?
    wait_for_completion=true&pretty" -H 'Content-Type: application/json' -d'
{
  "indices": "muj_index",
  "ignore_unavailable": true,
  "include_global_state": false
}'

dur=$SECONDS

echo "$(($dur)) seconds."
```

Listing 7.17: Vytvoření snapshotu v Elasticsearch

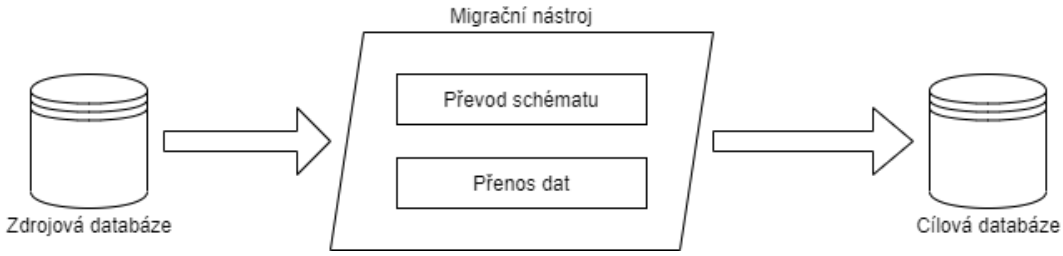
Zálohovaný Elasticsearch cluster lze obnovit následujícím HTTP požadavkem, ve kterém specifikujeme název snapshotu, ze kterého chceme provést obnovu a repozitář, kde je uložen.

```
#!/bin/bash

SECONDS=0

curl -X POST "localhost:9200/_snapshot/muj_repozitar/muj_snapshot01/_restore?
    wait_for_completion=true&pretty"

dur=$SECONDS

echo "$(($dur)) seconds."
```

Listing 7.18: Vytvoření snapshotu v Elasticsearch

7.3 Implementace nástroje pro migraci databází

Nyní se podíváme na implementaci nástroje pro migrování databází. Nástroj slouží k heterogenní migraci s kardinalitou 1:1 mezi databázovými systémy MySQL, MongoDB a Elasticsearch a také podporuje migraci do cloudových platforem AWS, MongoDB Atlas a Elastic Cloud a do Docker kontejnerů. Použitý migrační proces se dá znázornit následujícím diagramem.



Obrázek 7.5: Diagram migračního procesu

Lze z něho vyčíst, že implementovaný nástroj pro migraci provádí převod schématu databáze a přenos dat. Převod schématu se provádí pouze u heterogenní migrace, tedy například při přenosu databáze z technologie MySQL do MongoDB. U homogenní migrace se schéma znovu vytvoří v cílové databázi (MySQL -> MySQL), nebo v případě dokumentových databází, které schéma nemají, se tento krok přeskočí.

Nástroj se skládá ze čtyř skriptů v jazyce Python, kdy tři z nich slouží jako knihovny, které obsahují funkce pro migraci dané technologie. Nejprve se podíváme na funkce pro migraci databázového systému MySQL, které se nacházejí v souboru *migration_mysql.py*.

Migrace MySQL

Začněme migrací do jiné MySQL instance. Jak již bylo zmíněno, jako první je nutné vytvořit databázové schéma v cílové databázi. K tomu slouží funkce *showCreateTableSchemaMigration(sourceHost, sourceUser, sourcePass, sourceDb, destHost, destUser, destPass, destDb, destPort)*. Argumenty této funkce slouží k navázání spojení se zdrojovou a cílovou MySQL instancí. Jedná se tedy o adresy serverů (sourceHost a destHost), uživatelská jména a hesla (sourceUser, destUser a sourcePass, destPass), název zdrojové databáze (sourceDb) a cílové databáze (destDb). Proces přenosu databázového schématu vypadá následovně.

```
# Create and use new database in dest host
print(
    f"Creating database {destDb} in destination instance on {destHost}")
cmd = "create database if not exists " + destDb
cursor_dest.execute(cmd)
cmd = "use " + destDb
cursor_dest.execute(cmd)

# Turn off foreign key checks
cursor_dest.execute("SET FOREIGN_KEY_CHECKS=0")
```

```
    # Get tables from source database
    cmd = "select table_name " + \
        "from information_schema.tables " + \
        "where table_schema = \"" + sourceDb + "\" " + \
        "order by create_time asc"
    cursor_master.execute(cmd)
    tables = cursor_master.fetchall()

    # Create tables in dest database
    for tab in tables:
        tbl = tab[0]

        cmd = "SHOW CREATE TABLE " + tbl
        cursor_master.execute(cmd)
        create_table = cursor_master.fetchall()

        print(f"Creating table {tbl} on host {destHost}.")
        print(create_table[0][1])
        cursor_dest.execute(create_table[0][1])

    # Turn on foreign key checks
    cursor_dest.execute("SET FOREIGN_KEY_CHECKS=1")
```

Listing 7.19: Migrace databázového schématu

Nejprve se vytvoří v cílové instanci nová databáze a následně se vypne kontrola cizích klíčů. Díky tomu můžeme vytvářet tabulky v libovolném pořadí. Poté si načteme z databáze seznam tabulek a následně tyto tabulky vytvoříme. Nakonec znovu spustíme kontrolu cizích klíčů. Tímto způsobem lze migrovat databázové schéma z jedné instance do druhé.

Pro migraci dat do nově vytvořené databáze musíme pro jednotlivé tabulky nejprve sestrojit vkládací dotaz, pomocí kterého budeme jednotlivé záznamy vkládat. Na začátek si potřebujeme načíst seznam tabulek stejným způsobem, jaký je uveden ve výpisu 7.19. Poté si pro každou tabulku načteme popis její struktury (atributy) pomocí výrazu DESCRIBE a následně sestrojíme dotaz pro vložení záznamů.

```
# Get table description
cmd = "describe " + tbl
cursor_master.execute(cmd)
describtion = cursor_master.fetchall()
```

```
# Create insert query for table
cmd_insert = "insert into {} values(".format(tbl)
for i in range(len(describtion)):
cmd_insert = cmd_insert + "%s"
if i == len(describtion)-1:
    cmd_insert = cmd_insert + ")"
else:
    cmd_insert = cmd_insert + ","
```

Listing 7.20: Sestrojení dotazu insert into

Vytvořený vkládací dotaz pak můžeme použít pro migrování dat do tabulky v cílové databázi. Přenos se provádí následujícím cyklem:

```
print(f"Migrating data from table: {tbl}")
while(True):
    cmd = "select * from {} limit {}, {}".format(
              tbl, from_row, limit)
    cursor_master.execute(cmd)
    results = cursor_master.fetchall()

   # results set is empty => table migration is over
    if not results:
        break

   # insert data into table in target database
    cursor_dest.executemany(cmd_insert, (results))
    db_dest.commit()

    from_row = from_row + limit
    migrated = migrated + len(results)

print(f"Migrated {migrated} rows.")
```

Listing 7.21: Migrace dat do MySQL

Kromě výše uvedeného způsobu přenosu dat po dávkách lze migraci dat provést pomocí nástroje *mysqldump*. Tento způsob je ukázán v následujícím výpisu.

```
# Create new database in destination
```

```
print(f"Creating database {destDb} on destination instance {destHost}")
cmd = "CREATE DATABASE IF NOT EXISTS " + destDb
cursor_dest.execute(cmd)

# Execute mysqldump command in unix shell environment
print("Migrating data using mysqldump.")
cmd = f"mysqldump -u {sourceUser} -p{sourcePass} --single-transaction {sourceDb} |
    mysql -h {destHost} -u {destUser} -p{destPass} {destDb}"
os.system(cmd)
```

Listing 7.22: Migrace dat pomocí *mysqldump*

Proměnná *cmd* obsahuje příkaz pro spuštění nástroje *mysqldump* a přesměrování jeho standardního výstupu pomocí znaku "|"na standardní vstup cílové databáze. Příkaz je následně zaslán do unixového shellu pomocí funkce *system* z knihovny *os*. Přenos dat můžeme také provést pomocí exportu a importu CSV souborů, ale tento způsob lze použít pouze pro lokální migraci, jelikož server MySQL nepovoluje import dat ze souborů, které se nacházejí na vzdáleném serveru.

Migrace do NoSQL databází MongoDB a Elasticsearch probíhá téměř identicky. Liší se pouze v použitém databázovém konektoru a reprezentací tabulek. V MongoDB je každá tabulka reprezentována svou vlastní kolekcí v databázi a v systému Elasticsearch je pro každou tabulku vytvořen nový index. Konverze relační tabulky na dokument ve formátu JSON je pro oba systémy stejná a vypadá následovně.

```
# Get table description
cmd = "describe " + table
cursor_master.execute(cmd)
description = cursor_master.fetchall()

# Get atributes names
atrib_names = [x[0] for x in description]

# Get data from MySQL table
cmd = "select * from {} limit {}, {}".format(table, from_row, limit)
cursor_master.execute(cmd)
results = cursor_master.fetchall()

# Transform Sql data into JSON format
docs = []
for row in results:
    doc = {}
```

```
    for i in range(len(atrib_names)):
        doc[atr_names[i]] = row[i]
    docs.append(doc)
```

Listing 7.23: Konverze relační tabulky na JSON dokument

Prvním krokem je získání názvů atributů tabulky z výsledku dotazu DESCRIBE. Následně dochází k získání relačních dat a k vytvoření JSON dokumentu, který lze v jazyce Python reprezentovat datovým typem *dictionary*. Kolekce vytvořených slovníků(JSON dokumentů) pak lze jednoduše vložit do Elasticsearch indexu nebo MongoDB kolekce.

Migrování MySQL databáze do cloudové služby Amazon Web Services probíhá stejným způsobem, jako migrace do jiné instance MySQL serveru. V tomto případě se jako host udává tzv. endpoint, což je URL adresa služby. Poslední funkce v knihovně *migration_mysql.py* slouží k přenosu databáze do Docker kontejneru. Tato funkce vypadá následovně.

```
if not path.exists("mysql_docker/dump.sql"):
    print("File dump.sql does not exist. Please copy your mysqldump file into
        folder mysql_docker.")
    return

client = docker.from_env()
print("Creating docker image...")
client.images.build(path="mysql_docker", tag=imageName)
if not containerName == "":
    print("Running new container...")
    client.containers.run(imageName, name=containerName, detach=True)

print("Migration to docker done.")
```

Listing 7.24: Migrace MySQL databáze do Docker kontejneru

Pro přenos dat se používá sql soubor obsahující zálohu databáze. Nejprve dochází k inicializaci docker klienta, pomocí kterého následně vytvoříme docker image. Pokud dojde k úspěšnému vytvoření, tak se spustí nový kontejner, ve kterém běží naše databáze. Docker image se vytváří podle souboru *Dockerfile*, jehož obsahem je následující:

```
FROM mysql:latest as builder
RUN ["sed", "-i", "s/exec \"$@\"/echo \"not running $@\"/", "/usr/local/bin/docker
    -entrypoint.sh"]
ENV MYSQL_ROOT_PASSWORD=root
COPY ./dump.sql /docker-entrypoint-initdb.d/
```

```
RUN ["/usr/local/bin/docker-entrypoint.sh", "mysqld", "--datadir", "/initialized-
    db"]
FROM mysql:latest
COPY --from=builder /initialized-db /var/lib/mysql
```

Listing 7.25: Dockerfile pro MySQL

Dockerfile je velmi důležitý soubor pro vytvoření nového docker image, protože tímto souborem můžeme nastavit počáteční konfiguraci kontejneru včetně jeho obsahu.

Migrace MongoDB

Funkce pro migraci z databázového systému MongoDB se nacházejí v knihovně *migration_mongo.py*. Migrování do další instance MongoDB je snadná, protože nemusíme provádět žádnou konverzi dat ani nemusíme vytvářet databázové schéma, jelikož MongoDB je bezschémová databáze. Můžeme tedy provést následující přenos dat pomocí dávek následujícím způsobem.

```
while(True):
    # Get data from source
    results = sCol.find().skip(skip).limit(limit)
    data = list(results)

    if len(data) == 0:
        break
    # Insert data to target database
    dCol.insert_many(data)

    docs = docs + len(data)
    skip = skip + limit
print(f"Migrated {docs} documents.")
```

Listing 7.26: Migrace z MongoDB do MongoDB po dávkách

Migrovat můžeme také pomocí přesměrování výstupu nástroje *mongodump* do vstupu nástroje *mongorestore*. Celou migraci tak lze provést pomocí jediného řádku kódu. Funkce vypadá následovně.

```
def migrateMongoToMongoDump(sourceHost, sourceDb, destHost, destDb, destPort):
    print(
        f"Migrating {sourceDb} database on {sourceHost} to database {destDb} on {
            destHost}")
```

```
    cmd = f"""mongodump --host={sourceHost} --archive --db={sourceDb} |
        mongorestore --host={destHost} --port={destPort} --archive --db={destDb}"""
    os.system(cmd)
```

Listing 7.27: Migrace pomocí nástrojů *mongodump* a *mongorestore*

Funkce, která se stará o migrování dat do systému Elasticsearch pracuje podobně jako funkce pro migraci do MongoDB zmíněná ve výpisu 7.26. Rozdíl je samozřejmě v použitém konektoru pro komunikaci s Elasticsearch. Před vložením dokumentů musíme provést odstranění pole *_id*, které obsahuje objekt databáze MongoDB, který nelze vložit do Elasticu.

Pro přenos dat z MongoDB do relační databáze MySQL používáme v implementaci dva způsoby. První způsob, který je jednodušší, je vytvoření jedné tabulky, která obsahuje atribut s datovým typem *json*. Jeden záznam v tabulce tedy obsahuje jeden dokument ve formátu JSON. Postup vytvoření nové databáze, tabulky a přenosu dat je zobrazen v následujícím výpisu.

```
# Create database in dest host
cmd = "create database if not exists " + destDb
cursor_dest.execute(cmd)
cmd = "use " + destDb
cursor_dest.execute(cmd)

# Create table in new database
create_table = "create table if not exists cache (id int primary key
    auto_increment, mongo_cache json null)"
cursor_dest.execute(create_table)

# Insert json documents
insert = "insert into cache (mongo_cache) values (%s)"
for doc in docs:
    cursor_dest.execute(insert, (json.dumps(doc),))
```

Listing 7.28: Migrace do MySQL

Druhý způsob je složitější, protože vytváříme tabulku, která obsahuje atributy se stejným názvem a datovým typem, jako mají jednotlivá pole v dokumentu. K vytvoření tabulky slouží následující algoritmus.

```
create_table = "create table if not exists cache ( cache_id int primary key
    auto_increment, "
results = sCol.find_one() # Get 1 document
n_atrs = 0 # Number of attributes
```

```
# Loop through document fields
for i, key in enumerate(results):
    if(key == "_id"):
        continue
    value = results[key]
    create_table = create_table + key + " "
    insert_cmd = insert_cmd + key
    if type(value) == type(" "):
        create_table = create_table + "varchar(500)"
    if type(value) == type(1):
        create_table = create_table + "int"
    if type(value) == type([]):
        create_table = create_table + "json"
    if type(value) == type({}):
        create_table = create_table + "json"
    if type(value) == type(1.0):
        create_table = create_table + "float"
    if type(value) == type(True):
        create_table = create_table + "tinyint(1)"
    if i == len(results)-1:
        create_table = create_table + " )"
        insert_cmd = insert_cmd + " )"
    else:
        create_table = create_table + ", "
        insert_cmd = insert_cmd + ", "
    n_atrs = n_atrs + 1

cursor_dest.execute(create_table)
```

Listing 7.29: Vytvoření relační tabulky podle JSON dokumentu

V uvedeném kódu dochází k sestrojení dotazu pro vytvoření tabulky. K jeho sestrojení dochází v cyklu, ve kterém procházíme vzorový dokument a k proměnné *create_table* připojujeme řetězec s názvem a datovým typem pole procházeného dokumentu. Pomocí sestaveného dotazu následně vytvoříme tabulku. Poté nám stačí vytvořit vkládací dotaz stejným způsobem, jaký je uveden ve výpisu 7.20 a následně můžeme přenést data.

```
new_data = []
# Extract values from json document into list
```

```
for x in data:
    del x["_id"]
    lst = []
    for key in x:
        if(type(x[key]) == type([]) or type(x[key]) == type({})):
            lst.append(json.dumps(x[key]))
        else:
            lst.append(x[key])
    new_data.append(lst)
cursor_dest.executemany(insert_cmd, (new_data))
```

Listing 7.30: Přenos dat z MongoDB do MySQL

Migrace do cloudové služby MongoDB Atlas probíhá stejným způsobem jako migrace do běžné MongoDB instance, který je uveden ve výpisu 7.26. Pro připojení ke cloudové instanci se používá následující formát řetězce připojení:

```
mongodb+srv://{cloudUser}:{cloudPassword}@{cloudHost}/retryWrites=true&w=majority"
```

Listing 7.31: Connection string pro MongoDB Atlas

Také funkce pro migraci do Docker kontejneru je identická, používá pouze jiný Dockerfile pro vytvoření image. Ten vychází z nejnovější oficiální verze MongoDB. Během vytváření se do kontejneru nakopírují soubory s daty a skript *setup.sh*, který slouží pro naplnění databáze daty.

```
FROM mongo:latest
RUN mkdir /setup
COPY setup.sh /docker-entrypoint-initdb.d/
COPY . /setup
```

Listing 7.32: Dockerfile pro MongoDB

Migrace Elasticsearch

Veškeré funkce pro migraci systému Elasticsearch v knihovně *migration_elastic.py* jsou principiálně identické s těmi pro migraci MongoDB, pouze se používá knihovna *elasticsearch* pro připojení k databázi namísto *pymongo*. Důvod je jednoduchý, oba systémy pracují s dokumenty ve formátu JSON. Přenos dat do MongoDB nebo do jiné Elasticsearch instance probíhá tedy stejně jako v ukázce 7.26. Pro porovnání můžeme uvést ukázku funkce pro migraci do jiného Elasticsearch serveru.

```
# Get data from source index
cmd = {"from": offset, "size": limit, "query": {"match_all": {}}}
results = esSource.search(index=sourceIndex, body=cmd)['hits']['hits']
```

```
if not results:
    break

# Insert data into target index
helpers.bulk(esDest, results, index=destIndex)
```

Listing 7.33: Migrace z Elasticsearch do Elasticsearch

Pro migraci do relační databáze MySQL lze použít stejný postup konverze dokumentu na relační tabulky jaký je uveden ve výpisu 7.29. Podobné jsou i funkce pro migraci do Elastic cloudu a do Docker kontejneru. Ten má samozřejmě opět jiný Dockerfile pro vytvoření image.

7.4 Implementace nástroje pro navyšování výkonu

Třetí částí diplomové práce byla implementace nástroje, který by sloužil jako poradce pro navyšování výkonu databázového systému MySQL a implementace příkladů pro otestování výkonu. Nejprve se podíváme na implementaci těchto testů.

Účelem těchto testů je změřit výkon vykonání dotazů v databázovém systému před a po změně hodnot systémových proměnných, které jsme zmínili v kapitole 6.3. Jednotlivé testy se skládají ze dvou částí. První částí je funkce *run*, která slouží k zasílání dotazů na databázi. Ta pro otestování *innodb_buffer_pool_size* vypadá následovně.

```
def run_test_innodb(loops):
    # Connect to database
    db = mysql.connector.connect(
        host="localhost",
        user="martin",
        password="martin",
        database="dpl"
    )
    dbcursor = db.cursor()

    # Start sending queries
    for i in range(loops):
        limit = 1000
        offset = 20*i*1000
        q = f"SELECT * FROM produkty LIMIT {limit} OFFSET {offset}"
        dbcursor.execute(q)
        dbcursor.fetchall()
```

```
    dbcursor.close()
    db.close()
```

Listing 7.34: Funkce *run* pro testování InnoDB bufferu

Ve druhé části testu dochází k vytváření vláken, ve kterých běží výše zmíněná funkce *run*. Po dokončení tohoto vytížení se navýší hodnota testované proměnné a opět se spustí dotazování na databázi. Na tomto principu fungují všechny implementované testy.

```
threads = []
start = time.time()
# Create 8 threads running queries
for i in range(8):
    thread = mp.Process(target=run_test_innodb, args=(20,))
    threads.append(thread)
    thread.start()

# Wait for threads to finish
for t in threads:
    t.join()

end = time.time()
first_t = end - start
# Measure execution time
print(f"Execution time: {first_t:.2f}s")

# Increase variable value
dbcursor.execute("FLUSH TABLES")
new_buff = 4294967296
dbcursor.execute("SET GLOBAL innodb_buffer_pool_size = %s", (new_buff,))

# Start new threads here...
```

Listing 7.35: Funkce pro spuštění testování

Nyní se dostáváme k samotnému nástroji pro navyšování výkonu. Tento nástroj slouží jako poradce, který nám nabízí doporučené kroky pro zvýšení výkonu databáze. Skládá se ze dvou částí a to ze skriptu *mysql_helper.py* a knihovny *helper_functions.py*. Tato knihovna obsahuje funkce pro výpočet účinnosti InnoDB bufferu a key bufferu pro tabulky MyISAM, zjištění počtu fragmentovaných tabulek a výpočty poměru čtení a zápisu na databázi a velikosti požado-

vané operační paměti databáze pro aktuální konfiguraci. V následujícím výpisu je uvedena funkce *check_innodb_buffer_efficiency(cursor)*, která slouží k výpočtu účinnosti hlavního bufferu databázového engine InnoDB.

```
# Get number of reads
cursor.execute("SHOW GLOBAL STATUS LIKE 'Innodb_buffer_pool_reads'")
reads = cursor.fetchall()
# Get number of requests
cursor.execute("SHOW GLOBAL STATUS LIKE 'Innodb_buffer_pool_read_requests'")
requests = cursor.fetchall()

# Calculate efficiency as reads/requests*100
if (int(reads[0][1])/int(requests[0][1])*100) < 80:
    print("Done")
    print("[" + Fore.YELLOW + "Warning" + Style.RESET_ALL + "] " +
             "InnoDB buffer ma nizkou ucinnost, je vhodne navysit
                innodb_buffer_pool_size.")
    return True
else:
    print("Done")
    return False
```

Listing 7.36: Výpočet účinnosti InnoDB bufferu

Hlavní skript *mysql_helper.py* se skládá ze tří kroků. Prvním krokem je vyzvání uživatele k zadání přihlašovacích údajů k databázi, parametrů serveru a následné připojení k lokální instanci MySQL pomocí zadaných údajů. Zároveň probíhá kontrola zadaných informací, připojení k databázi a při jákemkoliv selhání je vypsána chybová hláška a skript je ukončen.

```
# Insert db username and password
db_user = input("Server username: ")
db_password = getpass.getpass("User password: ")
db_ram = input("Server RAM in GB: ")
db_cpu = input("Server CPUs: ")

# Validate input
if db_ram == "" or db_ram == " " or int(db_ram) <= 0:
    print("[" + Fore.RED + "Error" + Style.RESET_ALL + "] " +
          "Invalid value of server RAM.")
    quit()
```

```
if db_cpu == "" or db_cpu == " " or int(db_cpu) <= 0:
    print("[" + Fore.RED + "Error" + Style.RESET_ALL + "] " +
          "Invalid value of server CPUs.")
    quit()

db = None
# Try database connection
try:
    db = mysql.connector.connect(host="localhost",user=db_user,password=
        db_password)
except mysql.connector.Error as err:
    print("[" + Fore.RED + "Error" + Style.RESET_ALL + "] " +
          "Could not connect to localhost MySQL database.")
    quit()
```

Listing 7.37: Zadání a kontrola přihlašovacích údajů

Druhým krokem je načtení potřebných metrik a hodnot systémových proměnných z databáze a načtení seznamu "pravidel"ze souboru *rules.txt*. Tato pravidla obsahují název proměnné, jeji krajní hodnotu a text doporučené akce. Před načtením metrik a proměnných z databáze probíhá kontrola, zda máme dostatečná oprávnění a před otevřením souboru *rules.txt* se kontroluje, zda daný soubor existuje ve složce se skriptem.

```
thread_cache_size;Threads_created > 100;Vytvareni spousty novych vlaken, je
    potreba navysit thread_cache_size.
read_buffer_size;Select_scan > 2500;Vysoke pocty table scan operaci, je vhodne
    navysit read_buffer_size.
...
```

Listing 7.38: Obsah souboru *rules.txt*

V posledním kroku skriptu dochází jednak ke zpracování načtených pravidel, tak k volání jednotlivých funkcí z knihovny *helper_functions.py*. Zpracování pravidla se provádí následujícím algoritmem.

```
operace = {">": operator.gt, "<": operator.lt,
           "=": operator.eq, "<=": operator.le, ">=": operator.ge}

for line in rules:
    parts_of_rule = line.split(";")
    condition = parts_of_rule[1].split(' ')
```

```
if operace[condition[1]](int([st for st in status_vars if condition[0] in st
    ][0][1]), int(condition[2])):
    changes = changes + 1
    print("[" + Fore.YELLOW + "Warning" + Style.RESET_ALL + "] " +
          parts_of_rule[2])
    print("\n")
```

Listing 7.39: Zpracování pravidla ze souboru *rules.txt*

Shrnutí

V první části této kapitoly jsme se věnovali použitým technologiím pro implementaci praktické části diplomové práce. Jmenovitě se jednalo o použitý operační systém, programovací jazyky Python a Bash a jejich knihovny a databázové systémy, se kterými se pracovalo. Druhá část, která je věnována samotné implementaci, byla rozdělena na tři podkapitoly pro jednotlivé implementované nástroje a testovací příklady. V těchto podkapitolách jsme se věnovali popisu implementace a ukázkám zdrojového kódu.

Kapitola 8

Návrhy na zlepšení

V poslední kapitole diplomové práce se budeme věnovat návrhům na rozšíření implementovaných nástrojů o nové a zajímavé funkce. Řada existujících nástrojů, které řeší zálohování, migraci databází nebo navyšování výkonu, obsahuje spoustu funkcí, které zvyšují kvalitu nástroje a pohodlí uživatele. Cílem této kapitoly je tedy zmínit a popsat funkce, které by byly vhodným rozšířením pro námi implementované nástroje.

Grafické uživatelské rozhraní

Zřejmě nejzásadnějším vylepšením pro implementované nástroje by bylo rozšíření o grafické uživatelské rozhraní. Přestože má textové rozhraní pár výhod, jako je například jednoduchost nebo možnost spouštění na serverech, které mají většinou pouze shell, tak je grafické rozhraní lepší v mnoha ohledech. Jedná se zejména o možnosti využití různých grafických prvků, které jsou vhodnější pro zobrazování datových údajů, komunikace s uživatelem a poskytování příjemnějšího a intuitivnějšího prostředí pro uživatele.

Real-time monitoring metrik

Tato funkce by byla vhodným rozšířením nástroje pro navyšování výkonu. Cílem této funkce by bylo poskytování grafů s aktuálními hodnotami metrik databázového systému (počet aktuálních připojení, počet dotazů, využití bufferů apod.) a hostitelského počítače (zatížení sítě, CPU, operační paměti a disku). Díky těmto grafům by mohl uživatel snadno zjistit aktuální zatížení databáze a provést správné kroky k vyřešení výkonnostního problému. Hodnoty sledovaných metrik by se také ukládaly do databáze nebo log souborů pro pozdější analýzu.

Práce s dotazy

Další funkcionalitou, která by byla vhodná pro rozšíření nástroje pro navyšování výkonu, je práce s dotazy. Jednalo by se o funkci, která by monitorovala dotazy zasílané na databázi. Sledovala

by zejména četnost jednotlivých dotazů, jejich výkon a různé statistiky, jako je například velikost výsledku, počet logických a fyzických přístupů. Uživatel by tedy měl potřebné údaje k odhalení problémových dotazů.

Podporované databáze

Nástroj pro migraci databází by se dal zlepšit zejména z hlediska podporovaných databázových systémů. Z množiny relačních databází by bylo vhodné přidat podporu pro nejpoužívanější systémy, kterými jsou Oracle, SQL Server a PostgreSQL. Dále by se dal rozšířit o další typy databází, jako jsou například systémy key-value, grafové nebo XML.

Shrnutí

Ukázali jsme si tedy nové možnosti, jak zlepšit a rozšířit implementované nástroje. Zejména se jedná o vytvoření grafických rozhraní pro jednotlivé nástroje, real-time monitoring metrik a dotazů pro program, který řeší výkon databázového systému MySQL a přidání podpory dalších databází pro migraci. Zmínili jsme pouze pár možností rozšíření, ale dozajista existuje spousta dalších, které by se daly vymyslet a v následujících iteracích vývoje implementovat.

Kapitola 9

Závěr

Cílem této práce bylo seznámit se, analyzovat, zdokumentovat a otestovat možnosti navyšování výkonu, zálohování a migrace databázových systémů. Proto byla úvodní kapitola věnována analýze technologií kontejnerizace, virtualizace, cloudových služeb a CDN, které lze při řešení těchto problematik využít. V další kapitole jsme se věnovali enterprise edicím databázových systémů, zmínili jsme zejména jejich pokročilé funkce a nástroje, které ovlivňují kvalitu a výkon celého systému.

V kapitole 4 byly zpracovány teoretické znalosti o zálohování informačních systémů. Zmínili jsme obecné informace o zálohování, existujících nástrojích a replikaci vybraných databázových systémů. Následující kapitola projednávala o migraci databází, definovala pojem migrace a byly vysvětleny jednotlivé kroky migračního procesu. Kapitola 6 byla zaměřena na navyšování výkonu. Účelem této kapitoly je seznámení čtenáře s různými možnostmi navyšování výkonu relační databáze MySQL.

V kapitole 7 jsme se už zaměřili na praktickou část diplomové práce. Nejprve jsme se v podkapitole 7.1 seznámili s použitými technologiemi pro implementaci. Jmenovitě se jedná o použitý programovací jazyk Python a jeho knihovny, nástroj pro virtualizaci VirtualBox, cloudové služby AWS, MongoDB Atlas a databázové systémy MySQL, MongoDB a Elasticsearch. Následně jsme se věnovali vytváření úplných a inkrementálních záloh a procesu obnovy vybraných databází. Další podkapitola obsahuje popis nástroje pro migraci databázových systémů. V rámci této části bylo nutné najít dobré řešení pro převod rozdílných databázových schémat. Podkapitola 7.4 vysvětluje princip implementace nástroje, pro navyšování výkonu databázového systému.

Výsledná implementace dopadla dobře, což znamená, že se nám povedlo vytvořit funkční nástroje pro migraci databází a navyšování výkonu a funkční příklady pro vytváření záloh a obnovy databází. Je zde i samozřejmě prostor pro další vylepšení a rozšíření implementovaných nástrojů, kdy některé z nich jsou uvedeny v kapitole 8. V porovnání s existujícími projekty fungují implementované nástroje velmi podobně. Výhodou nástroje pro migraci databází oproti ostatním je možnost migrovat do Docker kontejneru. V případě nástroje pro navyšování výkonu se jedná o možnost úpravy konfiguračního souboru, podle kterého se vyhodnocují jednotlivá doporučení.

Literatura

1. *What is container* [online] [cit. 2020-10-05]. Dostupné z: https://www.docker.com/resources/what-container.
2. *Docker usage report* [online] [cit. 2020-10-05]. Dostupné z: https://sysdig.com/blog/2018-docker-usage-report/.
3. *Docker use cases* [online] [cit. 2020-10-05]. Dostupné z: https://www.docker.com/use-cases.
4. *Why you should use docker and containers* [online] [cit. 2020-10-05]. Dostupné z: https://www.infoworld.com/article/3310941/why-you-should-use-docker-and-containers.html.
5. *What is virtualization* [online] [cit. 2020-10-06]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-virtualization/.
6. *Virtualizace* [online] [cit. 2020-10-06]. Dostupné z: https://cs.wikipedia.org/wiki/Virtualizace.
7. *Co je to virtualizace* [online] [cit. 2020-10-06]. Dostupné z: https://www.inste.cz/2019/10/02/co-je-to-virtualizace-a-jak-dokaze-pomoci/.
8. *Virtualizace v kostce* [online] [cit. 2020-10-07]. Dostupné z: https://www.systemonline.cz/virtualizace/virtualizace-v-kostce.htm.
9. *What is the cloud* [online] [cit. 2020-10-07]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-the-cloud/.
10. *What is cloud computing* [online] [cit. 2020-10-07]. Dostupné z: https://www.oracle.com/cz/cloud/what-is-cloud-computing/.
11. *Cloud computing* [online] [cit. 2020-10-07]. Dostupné z: https://cs.wikipedia.org/wiki/Cloud_computing.
12. *Benefits of cloud computing* [online] [cit. 2020-10-07]. Dostupné z: https://www.ibm.com/cloud/learn/benefits-of-cloud-computing.
13. *What is a CND* [online] [cit. 2020-10-07]. Dostupné z: https://www.cloudflare.com/learning/cdn/what-is-a-cdn/.

14. *Content delivery network* [online] [cit. 2020-10-07]. Dostupné z: https://cs.wikipedia.org/wiki/Content_delivery_network.

15. *MySQL Reference Manual* [online] [cit. 2020-12-03]. Dostupné z: https://dev.mysql.com/doc/.

16. *MongoDB Enterprise Advanced* [online] [cit. 2020-12-05]. Dostupné z: https://www.mongodb.com/products/mongodb-enterprise-advanced.

17. *Elasticsearch Security* [online] [cit. 2020-12-06]. Dostupné z: https://www.elastic.co/security.

18. *Elasticsearch Features* [online] [cit. 2020-12-06]. Dostupné z: https://www.elastic.co/elasticsearch/features.

19. *Iperius backup tool* [online] [cit. 2021-03-06]. Dostupné z: https://www.iperiusbackup.com/backup-mysql.aspx.

20. *SQLBackupAndFTP backup tool* [online] [cit. 2021-03-06]. Dostupné z: https://sqlbackupandftp.com/.

21. *Percona XtraBackup* [online] [cit. 2021-03-06]. Dostupné z: https://www.percona.com/software/mysql-database/percona-xtrabackup.

22. *MongoDB Reference Manual* [online] [cit. 2020-11-16]. Dostupné z: https://docs.mongodb.com/manual/.

23. *Elasticsearch Guide* [online] [cit. 2020-11-19]. Dostupné z: https://www.elastic.co/guide/index.html.

24. *Database migration concepts and principles* [online] [cit. 2021-02-05]. Dostupné z: https://cloud.google.com/solutions/database-migration-concepts-principles-part-1.

25. ZHANG, Preston. *Practical Guide to Large Database Migration*. CRC Press, 2019. ISBN 978-1-1383-9162-8.

26. *What is database migration* [online] [cit. 2021-02-05]. Dostupné z: https://www.alooma.com/blog/what-is-database-migration.

27. *Database migration* [online] [cit. 2021-02-05]. Dostupné z: https://www.simform.com/database-migration/.

28. *MySQLTuner* [online] [cit. 2021-03-09]. Dostupné z: https://github.com/major/MySQLTuner-perl.

29. *MySQL Tuning Primer* [online] [cit. 2021-03-11]. Dostupné z: https://github.com/BMDan/tuning-primer.sh.

30. *Webyog Tuning Tool* [online] [cit. 2021-03-11]. Dostupné z: https://webyog.com/product/monyog/.

31. *Database Performance Analyzer* [online] [cit. 2021-03-11]. Dostupné z: `https://www.solarwinds.com/database-performance-analyzer/use-cases/mysql-performance-tuning`.

32. KRÁTKÝ, Michal; BAČA, Radim. *Databázové systémy*. 2020.

33. *Choosing Between SSD and HDD Storage* [online] [cit. 2021-03-11]. Dostupné z: `https://cloud.google.com/sql/docs/mysql/choosing-ssd-hdd`.

34. SCHWARTZ, Baron; ZAITSEV, Peter; TKACHENKO, Vadim. *High Performance MySQL, Third Edition*. O'Reilly Media, 2012. ISBN 978-1-449-31428-6.